UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Garaje Inteligente: Reconocimiento y Gestión de Matrículas de Vehículos.

Autor: Javier Fernández Santos
Tutor: Ana María González Marcos

junio 2020

Javier Fernández Santos
Garaje Inteligente: Reconocimiento y Gestión de Matrículas de Vehículos.

Javier Fernández Santos
C\Francisco Tomás y Valiente nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Agradecimientos

Escribo este apartado para dar por concluido mi trabajo de Fin de Grado y con ello poner fin a una de las etapas más especiales de mi vida. Una etapa fundamentada en el esfuerzo y dedicación, por las cuales he logrado realizar cosas como lo desarrollo de este trabajo, impensable e irrealizable antes de comenzar la carrera. Es por ello que me gustaría agradecer a todas las personas que me han apoyado durante toda esta etapa.

A mis padres y hermano, hacía los cuales solo puede expresar mi más sincero agradecimiento por todo su apoyo y esfuerzo, pues sin ellos esto no hubiera sido posible.

A mi tutora Ana, por darme la oportunidad de desarrollar este proyecto y por la orientación y ayuda que me ha brindado.

A mis amigos de la universidad, por el apoyo mutuo y por todos los buenos momentos vividos dentro y fuera de la universidad.

Gracias a todos.

Resumen

En este Trabajo Fin de Grado se va a desarrollar un **sistema encargado del control y administración de una red de garajes inteligentes y automatizados**. Para ello, el sistema estará **conformado por dos subsistemas**. Estos subsistemas son considerados independientes y deben funcionar de manera auto suficiente, compartiendo exclusivamente la información necesaria para dotar al sistema del objetivo inicial.

El primero de ellos consiste en el **reconocimiento automático de matrículas**. Es un método que deriva del campo de la visión artificial y está basado en la adquisición, procesamiento y análisis de las imágenes extraídas del mundo real para lograr su interpretación y comprensión. Para su desarrollo se ha hecho uso de un software opensource denominado OpenCV. El cual no solo proporciona métodos para el procesamiento y análisis de imágenes sino que también incorpora métodos para reconocimiento de objetos, entre otros. Con todo ello, se desarrolla un software en Python que comprende todas las fases necesarias para la obtención de la matrícula de un vehículo a partir de su imagen frontal o trasera.

El segundo subsistema consiste en el **desarrollo de una aplicación web**. Esta tiene como **propósito servir como red social** donde los usuarios puedan gestionar tanto sus garajes como las plazas de los mismos. Existen dos tipos de roles: usuario administrador y usuario personal. Cada uno de ellos dispone de una interfaz exclusiva que les permite la realización de acciones específicas en función de su rol. Por ejemplo, los usuarios personales podrán poner en alquiler sus plazas de garaje, mientras que los usuarios administradores tienen un histórico de los vehículos que han pasado por el garaje.
Su desarrollo está basado en el uso del framework Vue, el cual permite la creación de aplicaciones web basadas en la arquitectura SPA (Single Page Application). Por otro lado, se integran algunos de los servicios que aporta la plataforma para el desarrollo de aplicaciones denominada Firebase.

La **integración de estos dos subsistemas** permite la obtención de un sistema capaz de controlar los accesos a los garajes, basado en la extracción de las matrículas por medio del software de reconocimiento de las mismas y por los datos ingresados por los usuarios en la aplicación web.

Palabras clave

Reconocimiento de matrículas, ANPR, OpenCV, aplicación web, arquitectura SPA, Vue, Firebase

Abstract

In this end-of-degree Project, **a system will be developed to control and manage a network of intelligent and automated garages**. For this purpose, the system will **based on two subsystems**. These subsystems are considered to be independent and should operate in a self sufficient manner, sharing only the information necessary to provide the system with the initial objective.

The first of these consists of **automatic number plate recognition**. It is a method derived from the field of machine vision and is based on the acquisition, processing and analysis of images extracted from the real world to achieve their interpretation and understanding. For its development an opensource software called OpenCV has been used. It not only provides methods for processing and analyzing images but also incorporates methods for object recognition, among others. With all this, a software in Python is developed that includes all the necessary phases to obtain the license plate of a vehicle from its front or rear image.

The second subsystem consists of the **development of a web application**. This is intended to **serve as a social network** where users can manage both their garages and their parking spaces. There are two types of roles: administrator user and personal user. Each of them has an exclusive interface that allows them to perform specific actions according to their role. For example, personal users can rent out their garage spaces, while administrator users have a history of vehicles that have passed through the garage. Its development is based on the use of the Vue framework, which allows the creation of web applications based on the SPA (Single Page Application) architecture. On the other hand, some of the services provided by the platform for the development of applications called Firebase are integrated.

The integration of these two subsystems allows to obtain a system capable of controlling the accesses to the garages, based on the extraction of the license plates by means of the software of recognition of the same ones and by the data entered by the users in the web application.

Keywords

Automatic number plate recognition, ANPR, OpenCV, web application, SPA architecture, Vue, Firebase

Índice

Listas

Lista de figuras

1

Introducción

Este primer capítulo sirve como **sección introductoria** al Trabajo de Fin de Grado (TFG) , donde se explica la idea y el contenido del proyecto, la motivación que ha llevado tanto a su elección como a su desarrollo y una guía de la estructura del documento.

1.1. Descripción

Este TFG **se desarrolla en torno a dos temáticas.**

La primera de ellas es la ***Inteligencia Artificial***. Con la cual se aborda la problemática del **reconocimiento de matrículas de vehículos**. Para ello se recurre a la librería OpenCV con el propósito de desarrollar un programa en Python encargado de ello.

La segunda de ellas es el ***Desarrollo de aplicaciones Web***. Con la creación de una **red social para la gestión, monitorización y alquiler de plazas de garajes**. Para su desarrollo se recurre a Vue, un framework de javascript. Por otro lado, para no tratar con temas que no competen en el desarrollo de este TFG , se decide implementar la arquitectura Backend as a Service (BaaS) como modelo para el desarrollo de la aplicación web. Se escoge Firebase como plataforma para implementar dicha arquitectura.

En resumen, se desarrollan dos sistemas independientes que comparten información creando así una **red de garajes inteligentes y automatizados**.

1.2. Motivación

Año tras año surgen nuevas aplicaciones y herramientas que facilitan las tareas cotidianas y repetitivas. Este tipo de herramientas digitalizan la vida y poco a poco acaban desbancando la faceta tradicional de las cosas.

El principal motivo de la realización del proyecto es facilitar a la sociedad la gestión de garajes

por medio de una aplicación que incorpore una red social para el alta y la gestión de la información junto con el sistema de reconocimiento de matrículas. Si bien, existen por separado o integrados en el mismo sistema pero en un ámbito muy local como por ejemplo, el garaje de una empresa.

Por otro lado, el campo de la Inteligencia Artificial orientado al análisis y reconocimiento de objetos siempre me ha intrigado y con este proyecto tendré la oportunidad de indagar en él.

1.3. Objetivos

En este TFG se plantean **dos objetivos**. Aunque ambos se consideren sistemas independientes y deban funcionar auto suficientemente, deben compartir información para lograr dotar de inteligencia y automatización a la red de garajes.

El primero de ellos es el desarrollo de un programa en Python para el **reconocimiento de matrículas de vehículos**. Partiendo desde una foto de un vehículo, se deberá obtener como salida su matrícula, como se puede observar en la figura 1.1. Para ello es necesario realizar una serie de procedimientos:

1.– Acotar la región de interés, que en este caso es la matrícula.

2.– Segmentación de los caracteres extraídos de la zona de interés.

3.– Reconocimiento de caracteres.

El propósito no solo será integrar un software que incorpore técnicas capaces de realizar dichos pasos, sino también desarrollar métodos propios capaces de ello en base al estudio de técnicas ya existentes. Para ambos propósitos, se hará uso de la librería de visión artificial OpenCV, que aparte de incorporar procedimientos para el tratamiento y análisis de imágenes también posee técnicas avanzadas para los pasos citados anteriormente.

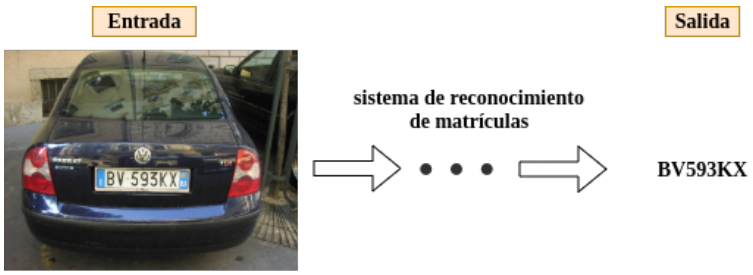

Figura 1.1: Ejemplo de entrada y salida en el reconocimiento de matrículas.

El segundo es el desarrollo de una **aplicación web**. El propósito es la **creación de una red social donde los usuarios puedan gestionar tanto sus garajes como sus plazas**. Se plantea que la aplicación sea sencilla de utilizar e intuitiva.
Dentro de la red social existen dos roles: *usuario administrador* y *usuario personal*. Cada uno de ellos dispone de una interfaz distinta que les permite realizar acciones específicas en función de su rol.

El *usuario administrador* representa una organización (comunidad de vecinos, trabajo, universidad, etc) la cual dispone de un garaje. Algunas de las vistas de la aplicación web relacionadas con el usuario administrador pueden verse en la figura 1.2. Se caracterizará por:

1.– Panel con el estado actual del garaje, es decir, plazas ocupadas y libres.

2.– Histórico de vehículos que han entrado y salido del garaje.

3.– Tramitar altas y bajas de usuarios personales dentro de la organización.

4.– Editar la asignación de plazas de garajes a los usuarios personales.

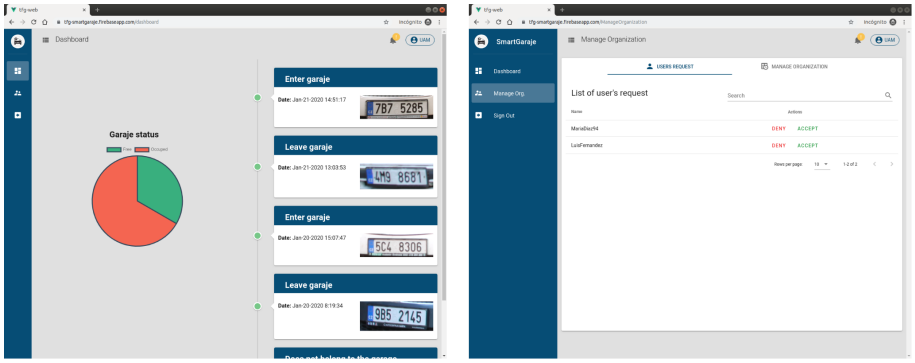

Figura 1.2: Ejemplo de interfaz para usuario administrador en la aplicación web.

El usuario personal representa una persona física. Algunas de las vistas de la aplicación web relacionadas con el usuario personal pueden verse en la figura 1.3. Se caracterizará por:

1.– Agregar, editar y borrar vehículos a su listado de vehículos. Con los cuales podrá acceder a los garajes.

2.– Solicitar acceso a una organización. Se podrá realizar por medio de un mapa interactivo o por búsqueda en una lista de organizaciones.

3.– Poner en alquiler plazas en desuso para que otros usuarios personales puedan acceder a ellas.

4.– Alquilar plazas de garajes de otros usuarios personales.

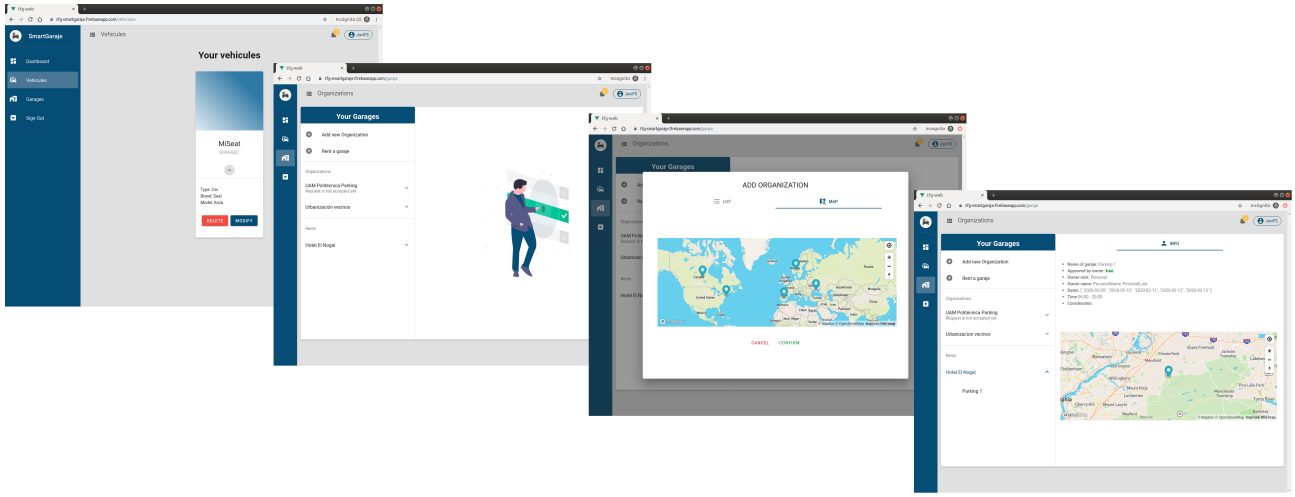

Figura 1.3: Ejemplo de interfaz para usuario personal en la aplicación web.

1.4. Estructura del documento

En este apartado se aborda la estructura que sigue este TFG . Se ha organizado en:

- En el ***Capítulo 1*** se realiza la introducción al TFG con una descripción del mismo y las motivaciones personales que han llevado a desarrollarlo. Por otro lado, se detallan las metas perseguidas en el proyecto.
- En el ***Capítulo 2*** se extraen de artículos, libros y tesis la información más relevante y vanguardista del estado actual tanto del sistema de reconocimiento de matrículas como de la aplicación web de gestión de los garajes. Es decir, se analiza el estado del arte para recopilar años de investigación y poder comenzar el trabajo con una base y mayor seguridad.
- En el ***Capítulo 3*** se recogen todas las actividades realizadas durante el análisis del TFG . El conjunto de actividades que se estudian son: alcance del proyecto seguido de la metodología y planificación del mismo. Se estudia el conjunto de tecnologías y herramientas necesarias para su desarrollo. Por último, se introducen los distintos roles presentes en la aplicación y los requisitos tanto funcionales como no funcionales que se deben satisfacer.
- En el ***Capítulo 4*** se expone el diseño que va a presentar el sistema en su totalidad. Por otro lado, se detalla la arquitectura de cada uno de los subsistemas presentes: sistema de reconocimiento de matrículas, aplicación web y base de datos.
- En el ***Capítulo 5*** se detalla el desarrollo de cada uno de los subsistemas poniendo en práctica los conceptos teóricos abordados en los capítulos anteriores.
- En el ***Capítulo 6*** incorporación de todas las pruebas realizadas y los resultados de las mismas.
- En el ***Capítulo 7*** se exponen las conclusiones a las que se ha llegado con el desarrollo y se abordan las posibles lineas de futuro trabajo.
- **Bibliografía:** recopilación de todas las referencias. Ordenadas por aparición en el documento.
- **Apéndices:** en el se incluyen diagramas, la interfaz y el flujo de la aplicación web, la instalación del SDK de Firebase y la creación de un clasificador Haar Cascade.

2

Estado del arte

Esta sección tiene como propósito analizar la evolución y el desarrollo de los campos involucrados en este TFG para así entender el contexto del cual se parte y dar enfoque al desarrollo del mismo. Para ello, se recopilan los precedentes y el estado actual de cada campo en cuestión para justificar el punto de partida y la elección de las tecnologías requeridas.

En primera instancia, nos adentramos en el mundo de la visión artificial para entender cuales son sus fundamentos. A continuación, el contenido se centra en el reconocimiento de matrículas de vehículos, que no es ni más ni menos que una problema derivado de la visión artificial.

Por último, se analizan los distintos enfoques y evoluciones que han vivido las arquitecturas web a lo largo de los años.

2.1. Visión artificial

Introducción

La vista es considerada como el mecanismo sensorial más importante en el ser humano. Es una actividad que permite analizar el entorno y se realiza de forma inconsciente. **La visión artificial es una disciplina que pretende trasladar esta capacidad innata de los seres humanos a las maquinas**, dotándolas no solo de visión, sino también con la capacidad de interpretación del entorno.
En el ser humano, el ojo es el encargado de capturar el entorno y la información del mismo es extraída e interpretada por el cerebro. Un sistema artificial se basa en el mismo proceso: la cámara hace de ojo, el software implementando actúa como cerebro para el procesamiento y análisis de la imagen.

Imagen digital

La cámara es el dispositivo encargado de capturar la información del entorno en un **formato procesable por el sistema**. Dicho formato es conocido como imagen digital.
Una imagen digital es una matriz de dos dimensiones (altura por anchura) donde cada elemento corresponde con un píxel. El píxel es la unidad mínima que conforma una imagen digital. Cada uno de

ellos se codifica con un conjunto determinado de bits para expresar la profundidad de color del mismo. En las imágenes reales se asignan 3 bytes (24 bits). Para poder definir el color de un píxel se necesita un modelo que lo defina, el modelo de color normalmente usado es el Red Green Blue (RGB) que permite crear cualquier color compuesto a partir de estos tres colores primarios. Es decir, según la cantidad de cada uno de ellos se logrará el color de cada píxel.

Un **aspecto crucial** para el buen funcionamiento del sistema de visión artificial, por lo menos para el reconocimiento de matrículas, **es la calidad que presenta la imagen**, como se puede comprobar en la figura 2.1. Para determinar la calidad de la misma se mide la densidad de píxeles por pulgada. Podemos determinar que a mayor densidad de píxeles, mayor precisión para la extracción de información del entorno.

Figura 2.1: Mitad izquierda con densidad de pixeles por pulgada (ppi) mayor que la mitad derecha.

Creación de un sistema de visión artificial

Todo sistema de visión artificial está fundamentado en una serie de fases [1]:

1.– **Adquisición de la imagen**.

2.– **Preprocesamiento:** la imagen digital que se adquirió puede sufrir en cierta medida algún efecto de degradación en forma de ruido. Esta fase pretende reparar aquellos desperfectos que han sido producidos por el hardware encargado de realizar la imagen. Para ello, se utilizan técnicas extraídas de [2] y [3].

3.– **Segmentación:** división de la imagen digital en zonas individualizadas. El objetivo es diferenciar los posibles objetos presentes en la escena aunque todavía no se sepa qué objetos son.

4.– **Análisis:** una vez se tienen acotados cada uno de los objetos, el análisis se encarga de obtener descriptores que definan las características de esos objetos: color, aproximación poligonal, dimensión, perímetro, etc. La clave de esta fase es saber seleccionar aquellos descriptores que sean discriminantes y permitan discernir el objeto.

2.2. Aplicación web

Introducción a la web

La web [4] o también conocida como World Wide Web (WWW) es uno de los muchos métodos que ofrece Internet para visitar páginas web conectados entre sí a través de enlaces. Estas páginas web están escritas en el lenguaje de marcado HyperText Markup Language (HTML) , por lo que para entender el contenido de las mismas es necesario disponer de un software, denominado **navegador web o cliente**, capaz de interpretar dicho lenguaje.
Por otro lado, se necesita que dichas páginas web se almacenen físicamente en algún lugar. A dicho lugar se le denomina **servidor**, que no es ni mas ni menos que un ordenador donde se almacenan y que tiene como propósito recuperar y entregar las páginas web requeridas por el cliente. Partiendo de esta introducción se pueden analizar las arquitecturas más usadas hoy en día.

Arquitecturas

Durante la evolución de la web han surgido diversas arquitecturas para el desarrollo de páginas webs. **No existe una arquitectura mejor que otra, sino que cada una está centrada para un uso en particular**.

La **arquitectura estática** [5] es considera como la más simple de todas, véase figura 2.2. En esta arquitectura solo se requiere de un servidor que aloje la página web. Como indica el nombre de la arquitectura, el contenido de dicha página web es estático por lo que no requieren más trabajo que la escritura del mismo. Esta arquitectura se usa para aquellos sitios webs orientados en la prestación de información estable a los visitantes. Por ejemplo, la web de nuestra escuela [6].
Como se dijo en la introducción, el cliente web se encarga de interpretar y renderizar la respuesta del servidor.

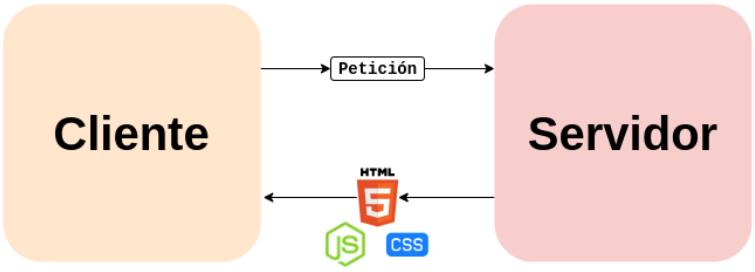

Figura 2.2: Arquitectura web estática.

No todas las páginas web están enfocadas para la presentación de información estable. De hecho, la gran mayoría de páginas web usan información muy cambiante e inestable. A medida que la página web crece en contenido, véase como ejemplo Amazon [7], su actualización es muy laborioso y hay que

recurrir a otra arquitectura. Se le denomina **arquitectura dinámica** [8].
La diferencia que presenta respecto a la estática es la presencia de una nueva pieza, una base de datos, véase en la figura 2.3. A partir de esta se obtiene todo el contenido que no es estable.

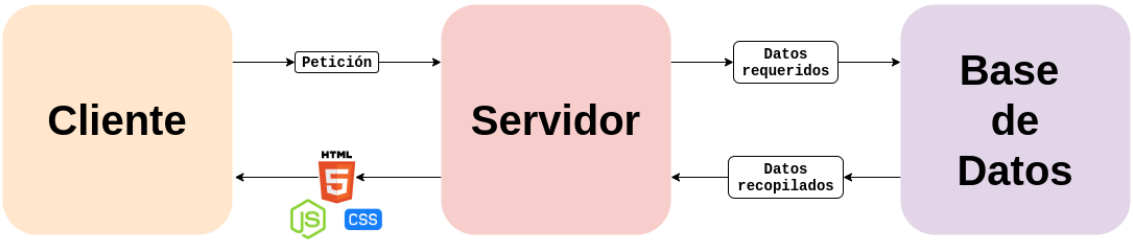

Figura 2.3: Arquitectura web dinámica.

En los últimos años ha surgido una nueva estructura que está cobrando mucha importancia en el sector web. Esto se debe principalmente a que ofrece una experiencia de usuario más agradable, al dotar a la página web con apariencia de una aplicación de escritorio, y el tiempo de espera se reduce considerablemente. La tendencia es tal que ha llevando a muchas empresas a actualizar su pagina web a esta arquitectura. Se está hablando de la **arquitectura Single Page Aplication (SPA)** [9].
Su arquitectura es muy similar a la arquitectura dinámica. La diferencia es que la arquitectura dinámica recarga todo el contenido presente, incluso aquellas partes que no han mutado. Mientras que la arquitectura SPA se basa en la actualización exclusivamente del contenido que ha mutado.
Como se observa en la figura 2.4, el cliente solo se comunica una vez con el servidor para obtener la estructura de la página, es decir, el contenido inmutable. Tras esto, el resto de comunicaciones se realizan contra la base de datos para recopilar la información que es requerida y que por tanto es mutable.

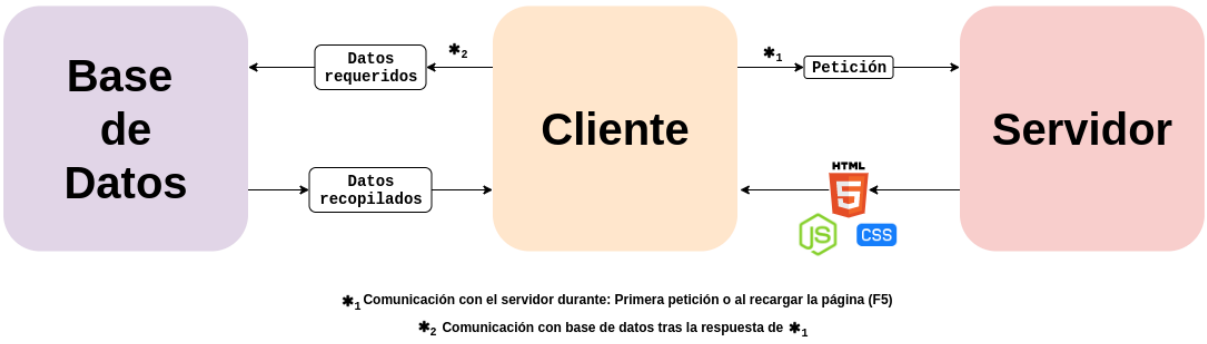

Figura 2.4: Arquitectura web SPA (Sinle Page Aplication).

3

Análisis y definición del proyecto

En esta sección del TFG se define el alcance del mismo, la metodología escogida, el conjunto de tecnologías y herramientas necesarias para su desarrollo, los distintos roles dentro de la aplicación y el conjunto de requisitos funcionales y no funcionales que debe cumplir el proyecto.

3.1. Alcance

Este TFG tiene como propósito desarrollar un sistema capaz de controlar y gestionar el acceso de vehículos a garajes. Para ello, el sistema estará formado por dos subsistemas. El primero será un software encargado del reconocimiento de matrículas de los vehículos que desean acceder al garaje. El segundo, una aplicación web que permitirá a usuarios y administradores gestionar el garaje.

Fuera del alcance de este TFG queda la implementación de la pasarela de pago en la aplicación web y la instalación del software de reconocimiento de matrículas en un miniordenador que se colocaría a la entrada de cada garaje, por ejemplo, la Raspberry Pi 3 podría servir, puesto que es lo suficientemente potente y permite la integración de módulos necesarios como una cámara de visión nocturna y un sensor de proximidad para encender la cámara.

3.2. Metodología

Para el desarrollo de este TFG , se optó por usar una **metodología de trabajo ágil**. Esto permite adoptar una estrategia de desarrollo incremental durante la cual se realizan las fases de análisis, diseño, codificación y pruebas. Al final de cada incremento, se dispone de un producto o prototipo listo para ser probado. Cuando se llega al último incremento, se obtiene el producto final.

En la figura 3.1 se muestra el **diagrama de Gantt** del proyecto. El objetivo del mismo es planificar el tiempo de dedicación previsto para cada una de las tareas a desarrollar.

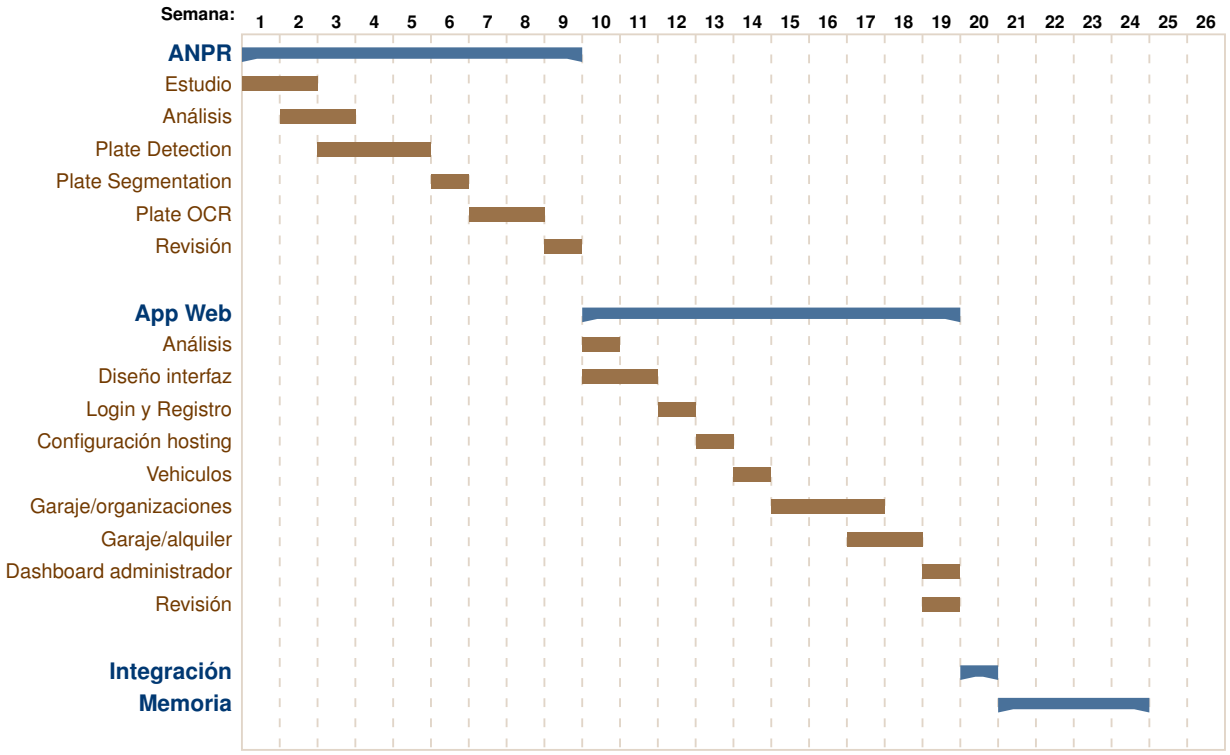

Figura 3.1: Planificación del proyecto mediante diagrama de Gantt.

3.3. Tecnologías y herramientas

Uno de los **aspectos más importantes en el desarrollo de un proyecto software es el estudio y análisis de las tecnologías y herramientas que más convienen en este**. Por esta razón, en esta sección se habla de las razones por las cuales se ha escogido cada una de ellas.

Para la implementación de este proyecto, se ha agrupado el uso de estas en **tres categorías**: *Reconocimiento de matrículas, Aplicación web y Gestión del proyecto*.

3.3.1. Reconocimiento de matrículas

Como se explicó con anterioridad en la sección 2.1, **las fases necesarias para la creación de un sistema de visión artificial requieren de**: obtención de un dataset completo con las imágenes necesarias y también un conjunto de tecnologías para el procesamiento, segmentación y análisis de las imágenes.

OpenCV

Entre las diversas tecnologías dedicadas al procesamiento de imágenes como scikit-image [10], mahotas [11], etc. Se ha decidido escoger OpenCV [12].

OpenCV es una biblioteca multiplataforma **enfocada al análisis y procesamiento de imágenes y vídeos que otorgan al sistema de visión artificial**. La elección de esta tecnología se debe a que es un proyecto opensource con dos décadas de desarrollo, dotando a la biblioteca con más de 500 funciones sobre procesamiento de imágenes, reconocimiento de objetos, etc. Por otra parte, su núcleo está escrito en C y C++, lo que permite una alta eficiencia en su ejecución.

Scikit-learn

Es una **librería de python para aprendizaje automático**. La elección de esta librería se debe a que incluye algoritmos de clasificación que serán utilizados durante el desarrollo del sistema de reconocimiento de matrículas.

Pytesserat

Es un **motor** Optical Character Recognition (OCR) **usado para el reconocimiento óptico de caracteres a partir de imágenes**.

Actualmente considerado como uno de los motores de reconocimiento de caracteres con más precisión. Google financia y da soporte a esta librería.

Imágenes de vehículos

Podría considerarse como la parte más importante, ya que es necesario disponer de un dataset completo para poder realizar de forma satisfactoria todas las fases del reconocimiento de matrículas. Fase bastante problemática debido a restricciones de uso de los datasets y mala calidad de imágenes. El dataset escogido [13] presenta una colección con gran variedad de vehículos e imágenes con perspectiva frontal o trasera del mismo.

Chars74k

Uno de los objetivos del proyecto es la creación de un software propio de reconocimiento de matriculas, esto requiere crear un motor OCR propio. Por esta razón, será necesaria un dataset para la fase de entrenamiento del mismo.

Chars74k [14] es una **colección que cuenta con más de 74.000 imágenes tanto de letras como de números**, divididos en tres formatos: caracteres manuscritos, generados por ordenador y extraídos

de fotografías de escenas cotidianas. Como las matrículas solo están formadas por números y letras mayúsculas con tipografía única y generada por ordenador se prescinde del resto de elementos de la colección. Un ejemplo de esta colección se muestra en la figura 3.2.

Figura 3.2: Ejemplo de la colección Chars74k para la letra P mayúscula generada por ordenador.

3.3.2. Aplicación web

En la sección 2.2 se introdujeron los tipos de arquitecturas más comunes en el desarrollo web. Para el desarrollo de este TFG se ha optado por implementar la arquitectura SPA , por lo que serán necesarias tecnologías que permiten su desarrollo. A parte, para dotar a la aplicación de toda la funcionalidad definida en los requisitos de la sección 3.5 se debe hacer uso de un conjunto de tecnologías que se describe a continuación.

Vue

Es un framework **de JavaScript para el desarrollo de páginas web basadas en la arquitectura** SPA . Caracterizado por ser una herramienta muy optimizada, permitiendo un desarrollo muy liviano. Posee una comunidad muy sólida que ha aportado librerías que pueden ser usadas para potenciarla.

Algunas de las librerías utilizadas son:

- **Vuetify** [15]: librería que provee herramientas para crear interfaces de usuario.
- **Vuex** [16]: Vue presenta un ecosistema basado en componentes. A medida que la aplicación crece, la dificultad para comunicar componentes también. Librería que facilita la comunicación entre componentes.
- **Vue-router** [17]: librería de enrutamiento.

Node.js

Es un **entorno de ejecución usado como capa de servidor**. Durante el desarrollo de la aplicación web se ha usado como servidor de desarrollo para comprobar los cambios que se iban realizando.

Firebase

Es una **plataforma que está basada en la arquitectura** BaaS , por lo que aporta un conjunto de herramientas orientadas al desarrollo de aplicaciones web y móviles que faciliten el desarrollo de las mismas. Los servicios usados de firebase [18] son:

- **Hosting**: como servidor de producción una vez finalizado el proyecto.
- **Realtime Database**: base de datos en tiempo real. Utiliza un patrón reactivo que permite la actualización automática de los datos en la aplicación.
- **Storage**: para el almacenamiento de imágenes.
- **Authentication**: servicio encargado de la identificación, registro y gestión de los usuarios.

MapBox

Es una **librería que permite la inserción de mapas dinámicos y personalizados** logrando su adaptación a la necesidad a tratar.

Google también dispone de una librería que aporta características similares (Google Map API [19]). Debido al reciente cambio en sus políticas de uso, es necesario ingresar una tarjeta de crédito para disponer de su funcionalidad. Por esta razón, se optó por el uso de MapBox [20].

3.3.3. Gestión del proyecto

Antes y durante el desarrollo de un producto software debe existir una metodología que permita planificar y guiar los procesos del proyecto. Permitiendo controlar y responder ante los problemas que surjan y facilitar la finalización del proyecto. Por esta razón, se ha hecho uso de las siguientes herramientas:

Trello

Es **software para la administración de proyectos que permite registrar las tareas por medio de post-it virtuales**. Esta herramienta es escogida porque permite implementar la metodología ágil scrum.

Git

Es una herramienta que sirve para el **control de las versiones de código fuente** que se van generando a lo largo del desarrollo del proyecto. Se utiliza GitHub como repositorio para almacenar el código fuente.

3.4. Roles

Antes de definir los requisitos que debe satisfacer el proyecto, se deben establecer los roles y el papel que va a desempeñar cada uno dentro del sistema. Existen dos roles:

Usuario personal: representa a una persona física. Tiene la capacidad de registrar los vehículos que más tarde usará en el acceso a cualquier organización, solicitud de acceso/-pertenencia a plazas de garaje y posibilidad de poner en alquiler aquellas plazas de garaje que le pertenezcan.

Usuario administrador: representa una organización (una comunidad de vecinos, una empresa, un garaje privado, etc). Encargado de tramitar las solicitudes que recibe de los usuarios personales. Tiene la capacidad de conocer en tiempo real la ocupación del garaje y un histórico de los vehículos que han pasado por él.

3.5. Análisis de requisitos

En este apartado se van a definir las características que debe cumplir el sistema tanto en funcionalidad (requisitos funcionales) como en usabilidad y calidad (requisitos no funcionales).

Requisitos aplicación web:

Requisitos funcionales:

RF-1.– Registro usuario - Independientemente del tipo de usuario, este debe indicar: un correo electrónico y una contraseña. Tras el registro del mismo, la cuenta no estará habilitada hasta que el usuario la active a través del mensaje que recibe en el correo electrónico introducido.

RF-2.– Registro usuario personal - Deberá contener: nombre, apellidos, alias y país del usuario.

RF-3.– Registro usuario administrador - Deberá contener: alias, el número de plazas que posee su garaje, la ubicación de la puerta del garaje en un mapa interactivo.

RF-4.– Inicio sesión usuario. Permitirá acceder al sistema por medio de correo electrónico y contraseña.

RF-5.– Usuario personal - Capacidad para agregar vehículos a su cuenta. Deben incluir una serie de propiedades: matrícula, tipo de vehículo, marca, modelo y dimensiones.

RF-6.– Usuario personal - Solicitar acceso a organización. Podrá hallar la organización filtrando su nombre en una tabla o a través de su ubicación en un mapa interactivo.

RF-7.– Usuario personal - Poner en alquiler plazas de garaje. Siempre y cuando la organización a la que pertenezca dicha plaza tenga habilitada la opción de alquiler. Seleccionará la plaza que quiera poner en alquiler, escogerá los días sobre un calendario interactivo y el horario de entrada y salida. Tras esta acción, la plaza de garaje quedará publicada y cualquier usuario personal podrá arrendarla.

RF-8.– Usuario personal - Alquiler de plaza publicada por otro usuario personal. Pudiendo escoger las fechas y horas que el propietario seleccionó.

RF-9.– Usuario administrador - Conocer el estado actual del garaje que gestiona. Por medio de un histórico que indica los vehículos que hayan entrado y salido, un indicador que muestra las plazas ocupadas y libres.

RF-10.– Usuario administrador - Gestión de las solicitudes de acceso enviadas por los usuarios personales.

RF-11.– Usuario administrador - Gestión del garaje. Podrá administrar las opciones de cada uno de los usuarios personales asociados al garaje.

Requisitos no funcionales:

RNF-1.– El sistema será desarrollado para plataformas web y móvil. Siendo capaz de adaptarse adecuadamente a la resolución del dispositivo.

RNF-2.– La interfaz de usuario será implementada para navegadores que soporten JavaScript.

RNF-3.– Hará uso de conexión a internet para su buen funcionamiento.

RNF-4.– No tardará más de 3 segundos en cargar las respuestas del servidor.

RNF-5.– Proporcionará mensajes de error y éxito informativos para el usuario.

RNF-6.– Las contraseñas han de estar protegidas por medio de una función hash y un valor aleatorio (salt) que eviten ataques de fuerza bruta producidos por diccionarios arcoíris.

Requisitos reconocimiento matrículas:

Requisitos funcionales:

RF-1.– Localización de matrícula - El sistema deberá ser capaz de ubicar la matrícula a partir de una imagen. Si se halla una matrícula, el sistema deberá generar una nueva imagen con el recorte de la misma.

RF-2.– Segmentación de caracteres - A partir de la imagen anterior, deberá ubicar cada una de los caracteres presentes en la matricula. Como salida, generará una imagen por cada carácter recortado.

RF-3.– Reconocimiento de caracteres - Analiza por separado cada carácter recibido del paso anterior.

Requisitos no funcionales:

RNF-1.– El tiempo de computo ha de ser inferior de 3 segundos.

RNF-2.– Debe ejecutarse en un entorno Linux y no podrá ocupar más de 500 MB de memoria RAM (en el trabajo futuro debería correr en una Raspberry pi 3).

4

Diseño

En este apartado se exponen los aspectos más relevantes en cuanto al diseño del sistema. En la primera sección se estudia la arquitectura que posee el sistema desde una perspectiva genérica. En las sucesivas secciones se trata en detalle cada una de las piezas que componen dicho sistema.

4.1. Arquitectura del sistema

Partiendo de una descripción de alto nivel del sistema, determinamos que está conformado por tres piezas, tal y como podemos ver en la figura 4.1:

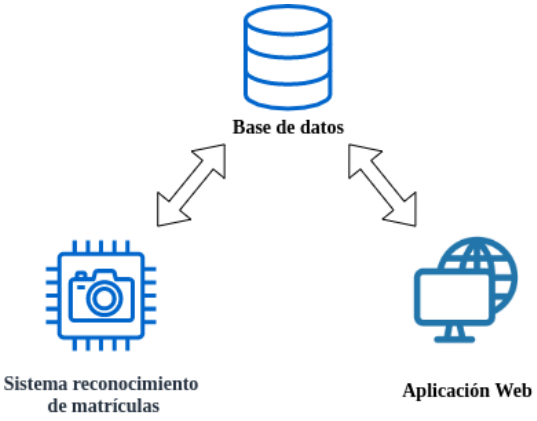

Figura 4.1: Arquitectura del sistema.

- **Sistema de reconocimiento de matrículas:** su tarea consiste en detectar la presencia de vehículos para realizar fotografías y comenzar con el análisis de las mismas. Pasados unos segundos y con la matrícula reconocida, se manda una petición a la base de datos para comprobar si dicha matrícula pertenece o no al garaje.
  Será necesario un miniordenador (ej. Raspberry Pi) en cada puerta de garaje.
- **Aplicación web:** desarrollada mediante el framework Vue. Esta pieza es la que permite a los usuarios administradores y personales gestionar las labores asociadas con los garajes.
  Durante la fase de desarrollo, se levanta un pequeño servidor en local que alojara la aplicación web. Una vez concluido el desarrollo, se opta por hacer la aplicación web accesible por medio del servicio de hosting que proporciona firebase.
- **Base de datos:** dispone de dos puntos de acceso. El primero es usado por el sistema de reconocimiento de

matrículas para consultar las matrículas analizadas. El segundo, usado por la aplicación web para el almacenamiento de todos los datos necesarios para el funcionamiento de la misma. Pieza alojada en el sistema de bases de datos de firebase que permite la sincronización de datos en tiempo real con una estructura NoSQL.

4.2. Arquitectura del sistema de reconocimiento de matrículas

Para crear un sistema capaz del reconocimiento de matrículas se ha hecho uso de la visión artificial. Siendo esta una disciplina científica derivada de la inteligencia artificial que incluye métodos para la adquisición, procesamiento, análisis y comprensión de imágenes. En definitiva, el propósito es el diseño de sistemas informáticos capaces de extraer información de una imagen tomada del mundo real.

En concreto, para resolver el reconocimiento de matrículas, será necesario que el sistema utilice técnicas de procesamiento de imágenes y de reconocimiento de patrones para extraer información sobre la matrícula presente en la imagen. Para ello es necesario aplicar cuatro fases:

1.– **Captura de la imagen:** esta fase es sustituida por imágenes de vehículos extraídas de la base de datos [13].

2.– **Localización de la matrícula:** fase considerada como la más compleja de todo el proceso de reconocimiento de la matrículas, pues entran en juego muchos objetos en la imagen que pueden ofuscar al sistema.
Como se puede observar en la figura 4.2, la fase parte con la imagen realizada al vehículo. A continuación, por medio de técnicas de procesamiento de imágenes se trata de extraer información útil para ubicar y acotar la matrícula del resto de la imagen. Para ello, se utilizan técnicas que recurren a la detección de extremos, búsqueda de coincidencias, extracción de punto de interés, etc. [21]
Para realizar esta fase ha sido necesario el uso de la librería OpenCV, de la cual se han **extraído técnicas de reconocimiento de objetos** como: Haar Cascade, Speeded-Up Robust Features (SURF) y Histogram of Oriented Gradients (HOG) .
Por otro lado, se implementa una **técnica propia llamada SelfMethod** basada en las nociones aprendidas durante la implementación de las técnicas extraídas de OpenCV.

Figura 4.2: Localización de la matrícula usando la técnica SelfMethod.

3.– **Segmentación de caracteres:** tras haber acotado la matrícula, se segmenta cada uno de los caracteres presentes en la matrícula, tal y como se observa en la figura 4.3.
Es considerada una fase crucial ya que un error durante la segmentación acarrea la incomprensión de la matrícula en la siguiente fase.
A pesar de ello, la complejidad para llevar a cabo el proceso es mínima. Por esa razón, se decidió implementar un método propio denominado SelfMethod. Para lograr esto, se vuelven a utilizar técnicas de procesamiento de imágenes extraídas de la librería OpenCV.
Otro aspecto a tener en cuenta durante esta fase es eliminar aquellos elementos indeseables de las matrículas como: guiones, puntos, país de la matrícula, etc.

Figura 4.3: Segmentación de cada carácter de la matrícula.

4.– **Reconocimiento de caracteres:** de la fase anterior se reciben tantas imágenes como caracteres haya en la matrícula. En cada una de las imágenes se ha de efectuar el mismo procedimiento. El primer paso es la binarización de la imagen, esto consiste en reducir la información presente de la imagen para normalizar el histograma y atenuar el ruido [22].
El siguiente paso consiste en la **extracción de características para su comparación con los patrones de caracteres ya definidos en el motor** OCR . De esta manera, el resultado será el carácter con el que más similitud tenga.
Algunas de las problemáticas que pueden surgir en esta fase están relacionadas con la calidad de la imagen. Entre las dificultades se pueden encontrar: iluminación pobre, imágenes desenfocadas, técnicas de evasión usadas por el propietario del vehículo, sombras, etc.
La salida que se obtiene al finalizar esta fase es la matricula formada por la unión de cada uno de los caracteres, como se puede observar en la figura 4.4
Para realizar esta fase se va a utilizar un motor OCR previamente entrenado como pytesseract y también se desarrolla un motor OCR propio para comparar resultados.

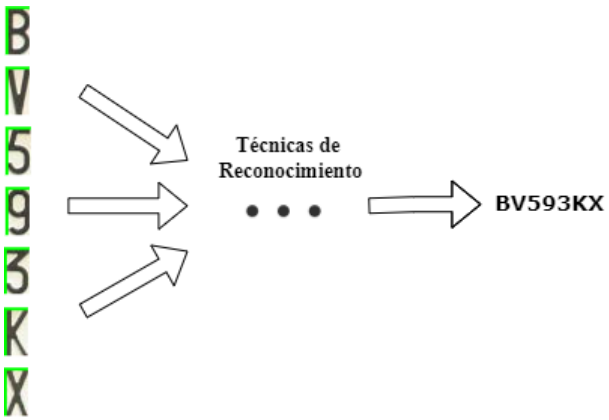

Figura 4.4: Clasificación individual de cada carácter.

4.3. Arquitectura de la Aplicación Web

Se ha escogido la **arquitectura BaaS** como modelo para el desarrollo de la aplicación web. Esta arquitectura se basa en otorgar al desarrollador una serie de servicios que le permita prescindir de la creación de una API personalizada.
La tecnología escogida para la implementación de dicha arquitectura es **Firebase**. Los servicios que se han integrado en nuestra aplicación son:

- **Base de datos en tiempo real:** permite prescindir de tareas como el mantenimiento y la optimización de la misma. El único contra es que debes adaptarte al ecosistema que ofrece.
- **Hosting:** permite alojar la aplicación en sus servidores.
- **Servicio de autenticación de usuarios:** para el control de acceso, registro y gestión de los usuarios.

En definitiva, **la elección de la arquitectura se basa en el hecho de no tratar temas que no competen en el desarrollo de este** TFG . De esta manera, liberamos al desarrollador de carga de trabajo en aspectos no trascendentales.
Esto da lugar a que la mayor parte del desarrollo se centre en la creación de la interfaz de usuario. La tecnología escogida para llevar el desarrollo de la interfaz es el framework de Javascript denominado **Vue**. Usado para el desarrollo de páginas web siguiendo el **patrón** Modelo, Vista, Vista-Modelo (MVVM) . La principal finalidad de este patrón es desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación.
Este patrón está compuesto por tres partes, tal y como se puede observar en la siguiente figura 4.5:

- **Modelo:** representa la capa con la lógica de datos y de negocio. En ella se contiene toda la información relativa a los objetos presentes en la aplicación, pero nunca incorpora las acciones que la manipulan.
- **Vista:** es la parte visible por el usuario, pudiendo este interactuar con la interfaz dando lugar a un flujo de eventos entre ella y la Vista-Modelo.
- **Vista-modelo:** se comporta como intermediario entre el Modelo y la Vista. Contiene la lógica de presentación y la definición de las acciones que puedan darse en la interfaz de usuario o Vista.

Figura 4.5: Patrón MVVM.

4.4. Arquitectura de la Base de datos

Alojada en el servicio de bases de datos de **Firebase**, el cual ofrece las siguientes características: escalabilidad, flexibilidad, almacenamiento en la nube y sincronización de los datos en tiempo real. Muchas de estas características son posibles debido al uso de estructuras de base de datos Not only SQL (NoSQL) .

Existen varios tipos de bases de datos NoSQL dependiendo de la forma en la que se almacena la información. En nuestro caso hacemos uso del tipo más versátil de todos: **base de datos documental**. En él, cada registro representa un documento con una estructura simple con formato JavaScript Object Notation (JSON) , permitiendo que el intercambio de datos sea ligero por la simplicidad de su formato.

En la figura B.2, apéndice B, se pueden observar los seis tipos de documentos presentes en la base de datos.

A continuación se hace una pequeña descripción de la información que recopila cada uno de los seis tipos de documento de la base de datos documental:

- **Usuario:** representa el rol de usuario personal. Está formado por el conjunto de vehículos que el usuario haya registrado. También contiene aquellas organizaciones donde haya sido aceptado tras el envío de la correspondiente invitación.
- **Vehículo:** contiene propiedades del vehículo. La más relevante es la matrícula, pues será la forma de validar el acceso al garaje de una organización.
- **Organización:** representa el rol de usuario administrador. Tiene una propiedad con las coordenadas para poder ubicar la puerta del garaje en un mapa. También dispone de una propiedad para habilitar o deshabilitar que los usuarios personales puedan alquilar sus plazas de garaje.
- **TimeLine:** representa cada uno de los movimientos que la cámara registra en una organización. Almacena la fecha, una foto de la matrícula capturada y el resultado final (entrada, salida, matrícula no reconocida o matrícula no perteneciente a la organización)
- **Plaza Garaje:** representa cada una de las plazas ubicadas en una organización. Cada plaza de garaje pertenece a un usuario personal.
- **Alquiler:** recopila toda la información necesaria para el alquiler de una plaza de garaje entre dos usuarios personales.

5

Desarrollo

En este apartado se expondrán los aspectos más relevantes sobre la puesta en marcha de los temas introducidos en las fases de Análisis y Diseño.

El desarrollo del proyecto se divide en tres etapas. La primera de ellas se enfoca en el desarrollo del software encargado del reconocimiento de matrículas. En segundo lugar, la creación de la aplicación web para la administración de garajes y usuarios. Y por último, se genera el enlace entre las dos etapas anteriores, de esta manera los datos recopilados por el software de reconocimiento de matrículas podrán ser usados en la aplicación web. Todo el código desarrollo durante este TFG se encuentra alojado en el repositorio [23] de GitHub.

5.1. Reconocimiento de matrículas

En esta primera sección se describen los pasos seguidos para el desarrollo del software reconocedor de matrículas. Solo se abordan los métodos de las librerías que mejores resultados generaron y se comparan con las propias implementaciones, denominadas 'SelfMethods'.

Las subsecciones tratadas son:

1.– **Localización de la matrícula**.
2.– **Segmentación de caracteres**.
3.– **Reconocimiento de caracteres**.

5.1.1. Localización de la matrícula

En esta subsección se tratan los dos métodos empleados, **Haar Cascade** y **SelfMethod**, para la localización de caracteres con sus correspondientes resultados. La fase parte con una imagen que puede o no contener una matrícula, la salida será una imagen con el recorte de la zona de interés.

Haar Cascade

Es un algoritmo **usado para la detección de objetos** en imágenes o vídeos basado en el concepto de 'características Haar' propuesto por Michael Jones y Paula Viola [24]. Es un **algoritmo de inteligencia artificial supervisado, ya que basa su lógica en imágenes previamente etiquetadas**.

Su funcionamiento es el siguiente: el algoritmo necesita un gran número de imágenes positivas (que contengan el objeto en cuestión) e imágenes negativas (que no contenga el objeto en cuestión) para entrenar el clasificador. Para ello, es necesario extraer características discriminantes que permitan determinar la presencia o ausencia del objeto. Esas características son conocidas como 'características Haar'. El valor de cada característica es la diferencia de la suma del valor de los píxeles en la zona blanca y negra de cada una de ellas. En la figura 5.1 se muestran ejemplos de ellas.

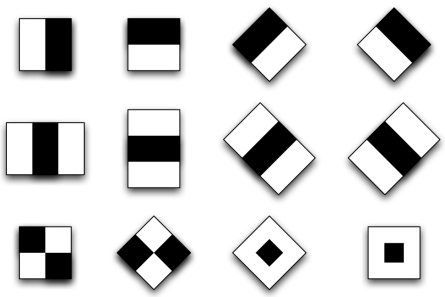

Figura 5.1: Algunos ejemplos de características Haar.

¿Cómo se aplican estas características? La idea es crear una ventana de un tamaño fijo que vaya recorriendo toda la imagen de forma solapada, como se puede apreciar en la figura 5.2. En el interior de esta ventana se calcula el valor cada una de las características. El problema que surge es que la gran mayoría de características que se aplican, pueden ser desde cientos a miles, son irrelevantes.

Figura 5.2: Ejemplos de ventanas en el método Haar Cascade.

¿Cómo se determina cuáles son relevantes? Por cada característica se busca el límite con el cual poder determinar la presencia o ausencia del objeto. De este proceso surgen características con menores tasas de fallo a la hora de clasificar, serán estas las características relevantes y las que

formarán parte del clasificador final.

Como se puede observar en la figura 5.3, el entrenamiento se realiza por etapas, en cada una de ellas se actualizan los pesos de las características generándose así un clasificador más robusto en cada nueva etapa.

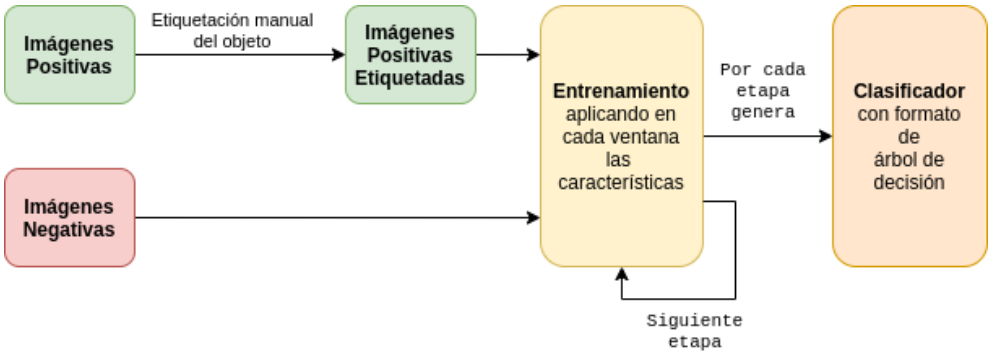

Figura 5.3: Fases para la creación de un clasificador Haar Cascade.

Cada característica relevante es considerada como un clasificador débil. La suma ponderada de todas ellas da lugar a un clasificador robusto. Se estiman necesarias cerca de 200 características relevantes para generar un clasificador con alta precisión (95 %), tal y como se afirma en el siguiente documento [25].

Terminado el proceso de entrenamiento y con el clasificador generado, es momento de probar su eficacia con nuevas imágenes y determinar el acierto tanto en casos positivos como negativos. El **proceso para clasificar una imagen consiste en**: recorrer toda la imagen aplicando las características relevantes en cada una de las ventanas y comparar los valores adquiridos con los almacenados en el clasificador, esta metodología está ilustrada en la figura 5.4. Aplicar en cada ventana tantas características es una tarea que requiere mucho tiempo. En vez de aplicar todo el conjunto de características en cada ventana, se introduce el concepto de Cascada de características. El cual consiste en generar un árbol de decisión de características. De esta manera, se comprueba una a una cada característica siempre y cuando la anterior no haya descartado la presencia del objeto, en dicho caso se pasaría a la siguiente ventana.

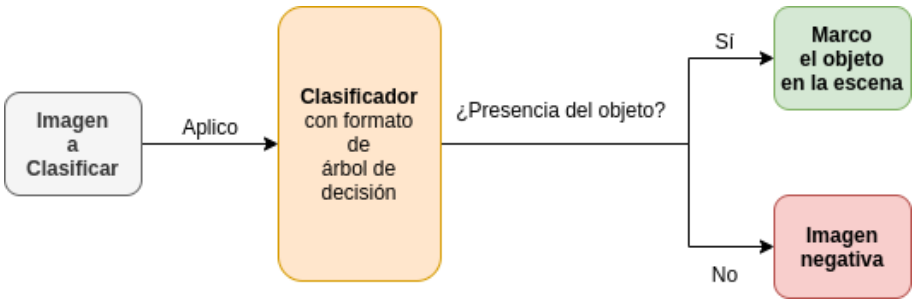

Figura 5.4: Procedimiento para clasificar una imagen mediante Haar Cascade.

Procedimiento:

Partiendo de lo anteriormente mencionado, **existen dos formas de proceder**:

1.– Realizar todos los pasos anteriormente mencionados para crear un clasificador Haar Cascade.

2.– Conseguir un clasificador Haar Cascade previamente entrenado y evitar así el primer paso.

Para el primero de ellos, se utilizan las herramientas implementadas por defecto en OpenCV. En el apéndice A se explica, con más en detalle, cual ha sido el procedimiento seguido para llevar a cabo esta primera opción. Para el segundo de ellos, se encuentra un clasificador ya entrenado [26] por los creadores de la librería OpenCV. La única información asociada a este clasificador es que ha sido entrenado durante 16 etapas con matrículas rusas.

Resultados y conclusiones:

Se recuerda que el objetivo es la localización tanto de matrículas rectangulares como cuadradas. Los resultados obtenidos tras aplicar la primera opción no son muy positivos debido al **alto tiempo requerido para el entrenamiento** del clasificador (8 horas para alcanzar seis etapas). Esto impide probar distintas configuraciones (distintos tamaños de ventana, un mayor número de etapas, mayor número de imágenes positivas, etc). Aún así, se llega a la conclusión de que Haar Cascade **no es un método eficaz si se tiene como objetivo la detección de ambos tipos de matrículas** (rectangulares y cuadradas) ya que no siguen un mismo patrón. Lo mejor sería crear un clasificador exclusivo para cada una de ellas y así evitar conflictos durante el entrenamiento, la figura 5.5 representa un posible esquema a seguir.

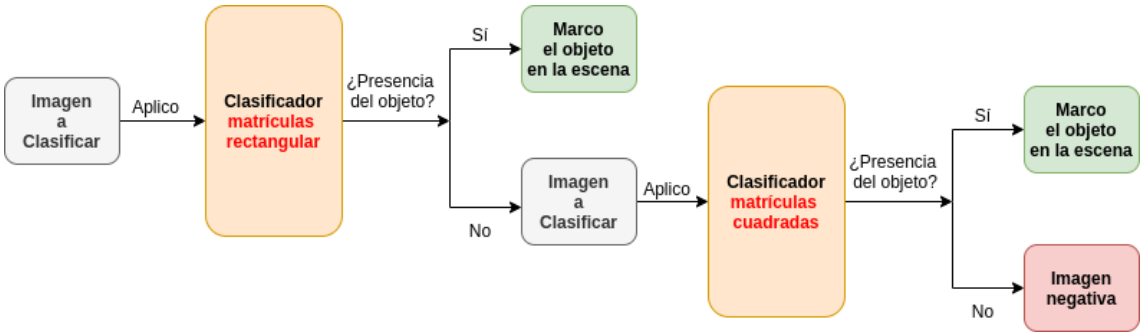

Figura 5.5: Posible esquema a seguir para la obtención de mejores resultados en la fase de clasificación de matrículas.

Por otro lado, el clasificador previamente entrenado con matrículas rusas tiene una **alta precisión para la localización de matrículas rectangulares**, mientras que las matrículas cuadradas no las localiza. Tras la investigación de [27] se descubre que en dicho país no existen matrículas cuadradas, por lo que es imposible que haya sido entrenado con ellas. Por esa razón no las detecta.

SelfMethod

Como se indicó en la introducción de esta sección, en cada una de las fases para el reconocimiento de matrículas se iba a realizar un método propio basado en la experiencia adquirida. Este en concreto se centra en la localización de matrículas, tanto rectangulares como cuadradas, a partir de una escena.

Es un **método que se basa en la morfología de los contornos extraídos de la escena para determinar si corresponden o no a algún tipo de matrícula**.
A pesar de ser un método que se centra en aspectos muy básicos ha dado muy buenos resultados.
Las fases quedan descritas en la siguiente figura 5.6.

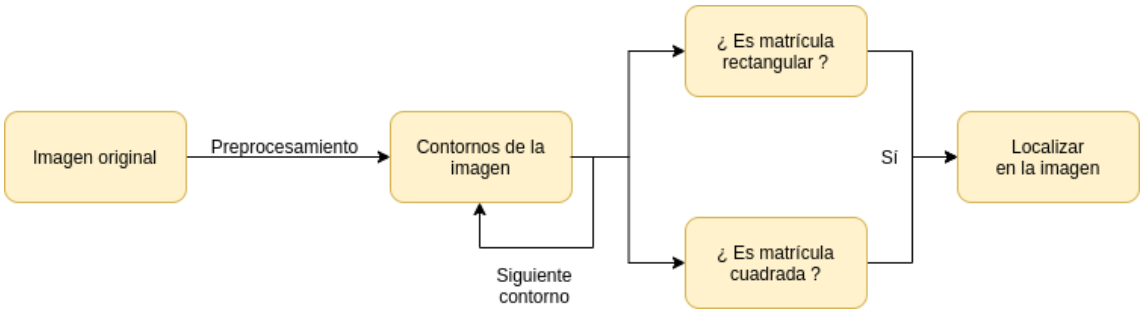

Figura 5.6: Fases para la localización de matrículas rectangulares y/o cuadradas basadas un algoritmo propio.

Como se ha indicado, este método se basa, fundamentalmente, en la morfología de los contornos extraídos de la escena. Pero **antes de obtener los contornos hay que realizar una serie de modificaciones sobre la imagen original para que esta nos los arroje**.
La imagen que se obtiene desde la cámara está formada por tres canales: rojo, verde y azul (24 bits). Lo primero que se debe hacer es transformar esta imagen a otra de un solo canal, 8 bits, obteniendo así una imagen en escala de grises. Con esta nueva imagen, se vuelve a transformar en otra representada exclusivamente por 1 bit. A este proceso se le denomina binarización, el cual consiste en dejar a cero todos los píxeles menores a un umbral y a uno aquellos iguales o superiores. Obteniendo así una imagen representada exclusivamente por ceros y unos (imagen en blanco y negro).
Con la imagen binarizada ya se pueden obtener de manera precisa los contornos presentes en la imagen.

Ahora que se dispone de todos los contornos de la imagen, **¿Cómo se determina si corresponde con una matrícula rectangular o cuadrada?**
La basta mayoría de países europeos [28] tiene una configuración similar en cuanto a matrícula rectangular, caracterizada por tener unas dimensiones de 520 mm de ancho por 110-130 mm de alto. De esta información se extrae la característica denominada **relación de aspecto (aspect ratio)**, que en este caso estará en torno a 4-4.7. En cuanto a las matrículas cuadradas no hay consenso y depende de cada país.

El dataset [13] con el que se está trabajando corresponde a matrículas croatas antes de su anexión a la Unión Europea. En aquel entonces la dimensión de la matrícula rectangular era de 520mm de ancho por 130-140 mm de alto. Por lo que se debe trabajar con ratios cercanos a 3.85. En cuanto a las matriculas que presentan un aspecto casi cuadrado, disponen de una dimensión de 230 mm de ancho por 160 mm de alto, con un aspect ratio de 1.4.
El aspect ratio no es la única propiedad que se aplica para la localización de las matrículas. También se hace uso de un método que permite comparar las **formas** entre dos contornos. Se proporcionan dos contornos predeterminados, rectangular y cuadrado, para ser comparados con cada contorno extraído de la imagen. Este método retorna un valor numérico que relaciona la similitud entre los contornos predeterminados y el extraído. Cuanto más cercano sea a cero, mayor similitud.
Otro aspecto relevante es el **área del contorno**. Si el área es nula quiere decir que no es un contorno cerrado por lo que no puede corresponder con el de una matrícula.
La última propiedad es el **perímetro del contorno**. Con él se representa la longitud del contorno. Si su valor es muy bajo o elevado, se descarta pues no corresponde a una matrícula.

Resultados y conclusiones:

En la figura 5.7 se muestran algunos de los pasos necesarios para la localización de matrículas rectangulares. Finalmente sobre la imagen original se remarca en color verde el contorno asociado con la matrícula detectada.

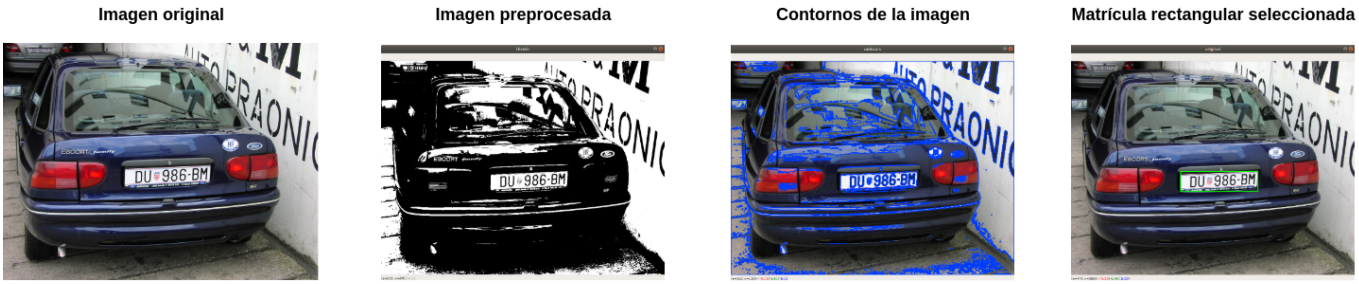

Figura 5.7: Localización de matrícula rectangular.

En la figura 5.8 se muestra la ejecución del programa 'Selfmethod' asociado con la figura anterior 5.7. En ella se puede apreciar que de la imagen original se extrajeron un total de 2133 contornos de los cuales solo uno presenta coincidencias con una matrícula de tipo rectangular. Esto se debe a que, como se puede apreciar en la figura 5.8, las propiedades aspect ratio y perimetro presentan valores aceptables y la comparación con el contorno rectangular predeterminado arroja un resultado muy bueno, pues es cercano a cero. Como los valores de estas características están dentro de lo estipulado en el código desarrollado, el contorno ha sido seleccionado como perteneciente a una matrícula rectangular, tal y como se puede apreciar en la última imagen de la figura 5.7.

```
python3 contours.py
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
[INFO] Total size of contorous list: 2133
[INFO] Possible plates:
 + Rectangle: aspect_ratio-> 3.875  perimeter-> 813.0782079696655  shape-> 0.20385104941655408
[INFO] Final size of contorous list: 1
```

Figura 5.8: Información que arroja el algoritmo propio tras su ejecución.

Se aplica el mismo proceso para matrículas cuadradas, como podemos ver en la figura 5.9.

Figura 5.9: Localización de matrícula cuadrada.

En este caso, como se puede observar en la figura 5.10, se han extraído un total de 645 contornos. De los cuales surge un candidato para matrícula rectangular y otro para cuadrada. Si se analizan los parámetros asociados con el candidato a matrícula cuadrada, se observa que presenta valores dentro de los límites establecidos, por lo que dicho contorno deja de ser candidato y pasa a ser considerado como matrícula cuadrada, como se puede apreciar en la útlima imagen de la figura 5.9. Si se analizan los valores asociados con el candidato a matrícula rectangular se puede observar que el aspect ratio no corresponde con el de una matrícula rectangular, por lo que este contorno queda descartado.

```
python3 contours.py
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
javi@javi  ~/Escritorio/TFG-ANPR/src/PlateDetection/SelfMethod  master  python3 contours.py
[VERSION CV]:  3.4.2
[INFO] Total size of contorous list: 645
[INFO] Possible plates:
 + Square:    aspect_ratio-> 1.4603174603174602  perimeter-> 301.11269783973694  shape-> 0.25396146979309
 + Rectangle: aspect_ratio-> 6.666666666666667  perimeter-> 462.5096662044525  shape-> 1.1240716305622267
[INFO] Final size of contorous list: 1
```

Figura 5.10: Información que arroja el algoritmo propio tras su ejecución.

Como conclusión es importante aclarar que **a pesar de ser un método fundamentado en aspectos muy básicos ha dado muy buenos resultados**. Debido a su simplicidad, es muy eficiente en cuanto a tiempo y recursos, ya que no precisa de entrenamiento alguno y las operaciones realiza-

das no suponen ningún esfuerzo a cualquier procesador actual, lo cual permite localizar la matrícula velozmente.

5.1.2. Segmentación de caracteres

La segmentación consiste en **dividir la zona de interés en zonas individualizadas**. El objetivo es la extracción de las características propias de cada objeto segmentado.
De la fase anterior, localización de la matrícula, se recibe la zona de interés. Como esta presenta un alto contraste entre los caracteres y el fondo de la misma su segmentación resulta muy sencilla. Por esta razón, en esta subsección se va a prescindir de cualquier método ya existente y solo se va a utilizar el método propio 'Selfmethod'.

SelfMethod

Con los buenos resultados obtenidos en la fase anterior con el método propio 'SelfMethod' basado en los contornos de la imagen, se decide aplicar la misma lógica para resolver esta fase.

Se **recopilan todos los contornos de la matrícula**. Al tratarse de una imagen acotada y que solo contiene información referente a la matrícula, da lugar a la existencia de un alto contraste entre los caracteres y el fondo de la misma, por lo que se obtiene un número muy limitado de contornos.

Para descartar aquellos contornos no pertenecientes a caracteres, es necesario **basarse en la altura y anchura que está impuesta por norma general**. La anchura máxima es de 45 mm y la mínima queda marcada por el carácter más estrecho (el uno). La altura mínima y máxima es de 77 mm. Con solo esta información es suficiente para segmentar los caracteres y evitar el resto de signos. El resultado obtenido es el que se muestra en la figura 5.11.

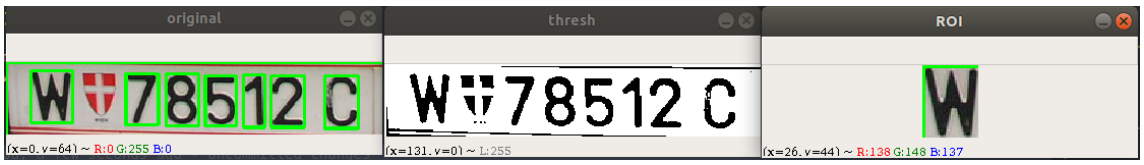

Figura 5.11: Segmentación de los caracteres de una matrícula que contiene un símbolo que podría ser confundido con un carácter.

5.1.3. Reconocimiento de caracteres

Es la última fase para dar por concluido el software reconocedor de matrículas de vehículos. De la fase anterior se recibe una imagen por cada carácter a reconocer ordenado de izquierda a derecha. El resultado final será la concatenación de la predicción de cada uno de ellos.

Pytesseract

Se recuerda que Pytesseract es un **motor** OCR desarrollado por Google para el reconocimiento de caracteres a partir de imágenes.

Procedimiento

Solo es necesario instalar la librería de Python para codificar un pequeño programa e ir comprobando las predicciones que determina este motor en cada imagen. Como no, también es necesario leer la documentación oficial de la librería para conocer su uso y funcionamiento.

Resultados

Los resultados que se han obtenido **no han sido tan buenos como se esperaba**. Esto se debe a que su motor ha sido entrenado para reconocer todo tipo de carácter posible de un idioma, como por ejemplo: letras, números, símbolos, etc. Por esta razón la precisión del clasificador se ve afectada.

SelfMethod

Al igual que en el resto de fases, en esta también se ha desarrollado un método propio para la fase en cuestión, el reconocimiento de caracteres. **Antes de entrar en los detalles de su desarrollo, se va a introducir el método de clasificación que se ha usado**.

Se ha utilizado un **método de clasificación supervisada llamado** K-Nearest Neighbors (KNN) . Es un método que estima la probabilidad a posteriori de que un elemento cualquiera pertenezca a una clase a partir de la información proporcionada por el conjunto de entrenamiento. Es decir, el algoritmo clasifica cada nuevo elemento según tenga un número (k) de vecinos más cerca de una clase que de otra. Se considera un método de aprendizaje vago puesto que todo el computo es diferido a la fase de clasificación.

La fase de entrenamiento consiste en almacenar cada elemento del dataset por: los vectores de atributos junto con su clase. Como se puede ver en el ejemplo de la figura 5.12, los elementos vienen representados por dos atributos y pueden pertenecer a dos clases (A o B). Una vez finalizada esta fase, todos los elementos estarán representados en una gráfica 2D (la dimensión depende del número de atributos con los que se representa cada elemento).

La fase de clasificación consiste en medir la distancia entre el elemento a clasificar y el resto de elementos adquiridos durante la fase de entrenamiento. Esta distancia representa la afinidad de atributos entre los elementos, existiendo una alta similitud cuando la distancia sea pequeña y alta desigualdad cuando la distancia es elevada. La esencia de la clasificación es **definir el número de vecinos** en los que se basa el clasificador para determinar la clase de un elemento (siempre ha de ser impar para evitar empates). En el ejemplo de la figura 5.12, se observa que un nuevo elemento (la

estrella) es clasificado como clase B según los 3 vecinos más cercanos. Sin embargo, es clasificado como clase A basándose en los 5 vecinos más próximos.

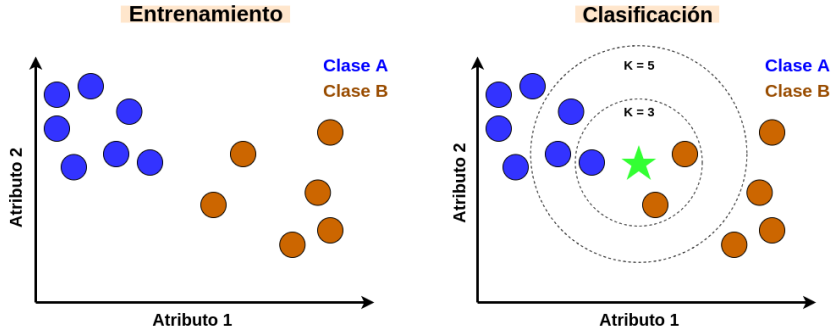

Figura 5.12: Ejemplo KNN.

Prerrequisitos

Es necesario un dataset con **imágenes de caracteres para la fase de entrenamiento**. Por ello, se utiliza la colección Chars74k [14]. Como se vio en la sección de tecnologías y herramientas, aunque el dataset esté formado por 74.000 imágenes solo son útiles los números y letras mayúsculas con tipografía única y generada por ordenador. Por cada carácter se dispone de 1000 imágenes para la fase de entrenamiento.

Procedimiento

Fase de entrenamiento: en el problema a resolver, los elementos corresponden con los posibles caracteres de las matrículas. Esto da lugar a **36 clases**: 10 para números y 26 para letras.
El procedimiento que se debe seguir para la obtención de un clasificador KNN se muestra en la siguiente figura 5.13.

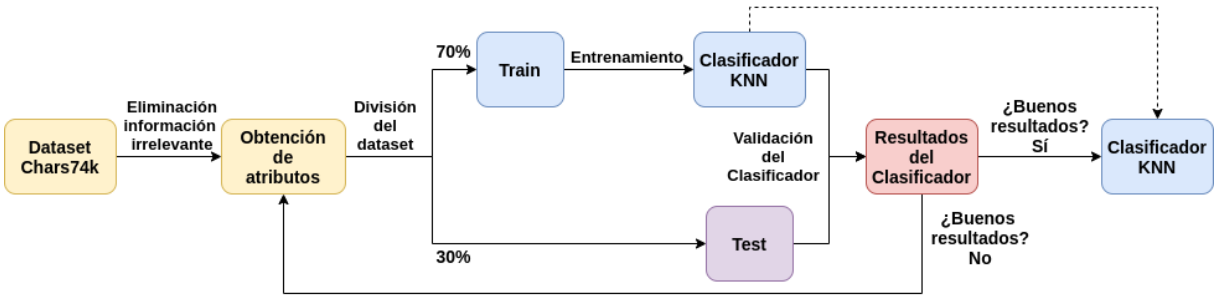

Figura 5.13: Procedimientos a realizar durante la fase de entrenamiento para la obtención del clasificador KNN.

La selección de atributos es la parte más compleja a la hora de aplicar el método de clasificación KNN, pues la elección de atributos no discriminantes da lugar a solapamiento de clases en el plano, lo cual imposibilita el uso de este método de clasificación. Con la elección adecuada de atributos se consigue que las clases queden diferenciadas entre sí, en el ejemplo de la figura anterior 5.12 se puede observar.

¿Qué atributos se van a seleccionar para representar cada carácter?

Para obtener los atributos se ha optado por dividir el carácter en recuadros. Se forma así una **rejilla** que divide el carácter en 6 recuadros de ancho por 6 de alto, lo cual da un total **de 36 atributos**, tal y como se puede observar en la figura 5.15. **Cada uno de los atributos va a contener la suma de los valores de los píxeles que queden en su interior**. Si todo el recuadro está formado por píxeles blancos (valor píxel blanco = 1), el valor de dicho atributo será su suma. Por el contrario, si todos son píxeles negros (valor píxel negro = 0), el valor de dicho atributo será nulo. La decisión de optar por estos atributos se debe a que los caracteres siempre van a tener una tipografía muy similar dando lugar a patrones muy característicos para cada carácter. Esto supone que cada carácter ocupará ciertas partes de la imagen mientras que otras no.

Antes de aplicar el sistema de rejilla sobre la letra, es necesario eliminar información irrelevante de la misma. Por ejemplo, en la figura 5.14 se muestra una letra 'A' extraída del dataset Chars74k que va a ser usada para crear el modelo KNN, se pueden observar zonas blancas que rodean la letra.

Si se recuerda del capítulo 2, estado del arte, una imagen está representada por una matriz de píxeles. Para eliminar las zonas carentes de información, se va a eliminar aquellas filas de píxeles que estén formadas en un 95 % por píxeles blancos. Si se aplica esta regla, se puede observar como entre la ventana (1.1) y (1.2) de la figura 5.14 han desaparecido las zonas blancas que quedan por encima y debajo de la letra. Entre el paso (2.1) y (2.2), antes de aplicar lo anterior, se realiza la transpuesta de la matriz de píxeles para rotar la imagen. En la ventana (2.2) se observa como ha desaparecido el contorno blanco de la imagen original. El único problema es que no presenta la orientación adecuada, por lo que se vuelve a realizar la traspuesta para dejar el carácter sin contorno blanco y en su posición original, como se puede ver en la ventana (3) de la figura 5.14.

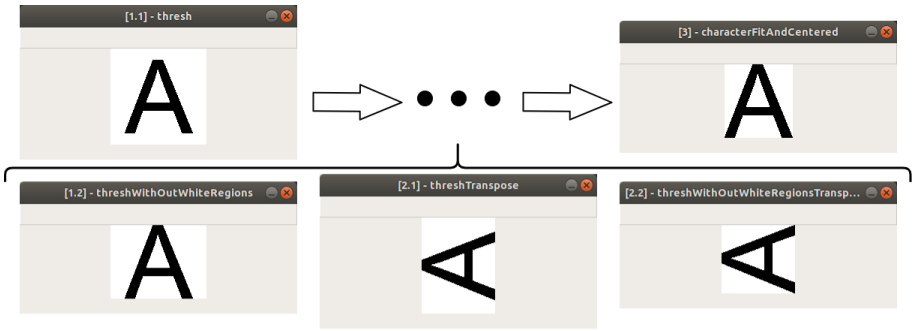

Figura 5.14: Eliminación de información irrelevante.

Haber eliminado información irrelevante origina un problema. Antes de ese paso todas las imágenes extraídas del dataset chars74k tenían la misma resolución. Ahora cada imagen va a tener una resolución distinta, lo que provoca que los atributos no sean del mismo tamaño. Por ejemplo, la letra 'A' de la ventana (3) de la figura 5.14 queda con una resolución de 91 píxeles (px) de ancho por 98 px de alto. Si se aplica el sistema de rejilla, tal y como se puede observar en la figura 5.15, no todos los atributos presentan la misma resolución. Si nos fijamos en la primera fila desde arriba, desde la primera a la quinta columna incluidas, los atributos presentan una superficie de 240 px cuadrados (15 px de ancho por 16 px de alto), mientras que en la última columna presenta una superficie de 256 px cuadrados (16 px de ancho por 16 px de alto). Como no todos los atributos tienen la misma resolución, **el valor de cada atributo será la media de los valores de los píxeles contenidos en su interior**.

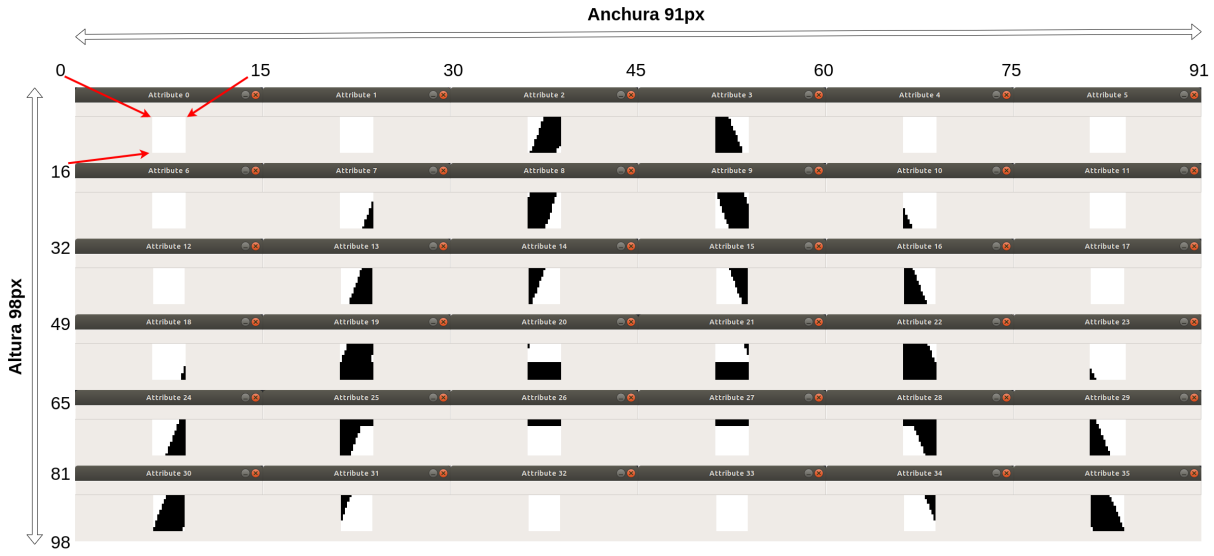

Figura 5.15: Segmentación de cada uno de los 36 atributos. Ejemplo de una letra 'A' cualquiera.

Una operación común en los modelos de aprendizaje supervisado es la **división de nuestro conjunto de datos** (el dataset chars74k) en dos partes: una parte **'Train'** (70 %), que se usará para entrenar nuestro modelo, y un parte **'Test'** (30 %), sobre la que se evalúa el modelo entrenado.
El dataset chars74k dispone de casi 1000 imágenes por carácter. Por lo que el modelo será entrenado con 700 imágenes por carácter y será validado con 300 imágenes por carácter.

Una vez realizada la fase de entrenamiento se obtiene el clasificador. Antes de tomarse como válido y ser usado con datos de producción, se **debe validar el clasificador** con los datos 'Test', tal y como se ve en el esquema de la figura 5.13.
Se va variando el valor de los vecinos (k) para ver cual de ellos da un mejor resultado. Se puede observar en la tabla 5.1 que a medida que se escoge un número mayor de vecinos la tasa de acierto disminuye y el tiempo de computo es mayor, como era de esperar pues ha de realizar más comparaciones.
Con un vecino (k=1) se observa que la tasa de acierto es de casi un 96 %. Con esta cifra se pue-

de afirmar que los atributos escogidos durante la fase de entrenamiento fueron una buena elección y suficientemente discriminantes, pues han logrado crear un clasificador muy preciso.

K vecinos	Aciertos	Fallos	Precisión	Tiempo ejecución (seg)
1	6241	268	95.88 %	2.88
3	6086	423	93.5 %	4.73
5	6074	435	93.31 %	5.62
7	6068	441	93.22 %	6.46
15	6021	488	92.5 %	7.61
21	5978	531	91.84 %	8.8
51	5875	634	90.25 %	10.88

Tabla 5.1: Resultados de la fase de validación aplicando distinto número de vecinos..

Fase de clasificación: Una vez que el clasificador obtiene buenos resultados de validación durante la fase de entrenamiento se puede empezar a utilizar con datos reales.

En este caso, los datos reales corresponderán a los caracteres extraídos desde una matrícula. Antes de probar con estos datos, es necesario tratarlos de la misma manera que se hizo con los datos del clasificador, esto es: preprocesar la imagen, eliminación de información irrelevante y extracción de los atributos. Hecho esto, el clasificador devolverá una predicción basada en los atributos extraídos de cada imagen, tal y como se muestra en la figura 5.16.

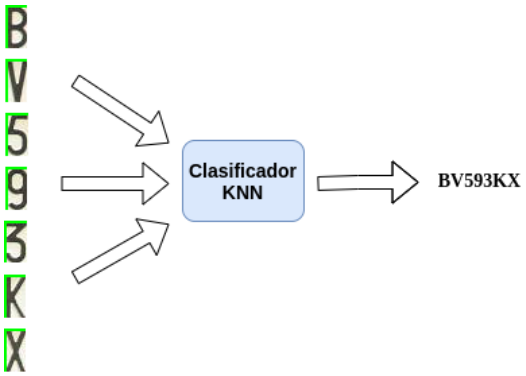

Figura 5.16: Clasificación de los caracteres de una matrícula con el modelo generado durante la fase de entrenamiento.

Resultados

Antes del desarrollado del clasificador se conocían las **dificultades** que se iban a presentar. Si bien es cierto que el hecho de tratar con caracteres con tipografía única disminuye la complejidad del desarrollo, existen ciertos caracteres que presentan una alta tasa de coincidencia que dificulta la predicción. Estos caracteres son: la 'O' con el cero, la 'Z' con el 2 y la 'B' con el ocho, como se puede observar en la figura 5.17.

Figura 5.17: Dificultades durante la fase de entrenamiento y clasificación.

A pesar de estas dificultades, los **resultados obtenidos son mejores de lo esperado**. La única flaqueza que presenta el clasificador es al clasificar una 'O' o un cero, pues la precisión que tiene en este caso es la misma que lanzar una moneda. De todas maneras, este problema tiene fácil solución en matrículas que agrupan números o letras por secciones, por ejemplo, en España las matrículas más modernas comienzan con cuatro números seguidos de tres letras. El clasificador puede determinar si es una 'O' o un cero en función de la posición del carácter, si se presenta en el grupo de los números será el cero y si se presenta en el de letras será la 'O'.

5.2. Aplicación web

Esta sección corresponde a la segunda parte de la fase de desarrollo. Está enfocada al desarrollo de la aplicación web donde los usuarios y administradores pueden realizar todas las acciones contempladas en los **casos de uso** de la figura B.1 situada en el apéndice B. Por otro lado, en el apéndice C, se muestra **la interfaz que tiene la aplicación web y el flujo de la misma**.

5.2.1. Frontend

También conocida como capa de presentación, en ella **se ha desarrollado la lógica encargada de servir la interfaz para el usuario** y que él mismo pueda interactuar con ella. Se ha hecho uso del framework de JavaScript denominado **Vue** [29] para la creación de una aplicación web basada en una arquitectura SPA .

Antes de comenzar el desarrollo de la aplicación web se realizó un diseño previo para la creación de una plantilla que sirviera como base a toda la aplicación. Para ello, se hizo uso del paquete **Vuetify** [15], encargado de proveer esqueletos de componentes de interfaz a los que simplemente fue necesario dotar de la funcionalidad requerida en cada caso.

Para lograr que la aplicación web se comporte como una aplicación de una sola página SPA es necesario la integración del **paquete de enrutamiento** que es proporcionado de manera opcional por Vue. Este paquete es conocido como **vue-router** [17] y es el encargado de la gestión interna de las rutas de la aplicación web. Una de las características más importantes que provee es la capacidad de proteger las rutas, que para el proyecto fue de gran interés pues permite controlar que administradores y usuarios personales navegan por las rutas que les corresponden. Es decir, prohíbe que los usuarios personales naveguen por rutas del administrador, y viceversa.

La idea principal de la aplicación web es que los usuarios personales puedan acceder a dos tipos de garajes: los promocionados por un usuario administrador (empresas, comunidades de vecinos, universidades, etc), y por otro lado y ajeno al anterior, los garajes promocionados por los propios usuarios personales, es decir, personas particulares que quieran poner en alquiler su plaza.
Para lograr esto y que fuera intuitivo para el usuario se decide hacer uso de **mapas interactivos**. Para ello, se utiliza el Software Development Kit (SDK) [30] de **MapBox** [20] para incrustar un mapa personalizable en el cual las chinchetas representan cada uno de los garajes.

5.2.2. Backend

Como se explicó con anterioridad, para no extender el desarrollo del TFG se decidió optar por la implementación de la arquitectura BaaS , **la cual permite contratar los servicios de un tercero para**

la adquisición de la funcionalidad requerida en la parte del backend. Se recurre a la herramienta que proporciona Google denomina Firebase [18], que con su plan gratuito permite disponer de todo lo necesario para este proyecto. Más información sobre la integración de este servicio en el apéndice D.

Para **controlar el acceso y registro de usuarios** a la aplicación se hace uso de 'Firebase Authentication', un sistema que facilita la creación de un sistema seguro de autenticación. En este proyecto la autencicación está basada en el uso de un correo electrónico y una contraseña. Un aspecto muy importante a la hora de desarrollar un producto software es la **protección de información sensible de nuestro usuarios**. Firebase proporciona de forma automática esta protección, las contraseñas no se almacenan en texto plano, sino que aplican una función hash a la contraseña junto con un salt, complicando la decodificación de las contraseñas en caso de que se hiciera publica la base de datos del sistema de autenticación.

Firebase también proporciona una **base de datos** NoSQL para el almacenamiento, sincronización y consulta rápida y eficiente de los datos necesarios para el funcionamiento de la aplicación. Está basada en el uso de colecciones y documentos para estructurar los datos con facilidad.
También provee de un sistema denominado 'Firebase Storage' que sirve como almacén para el **alojamiento de imágenes**. De esta manera, podremos almacenar y procesar con rapidez el contenido generado por los usuarios. En este caso, las imágenes realizadas por el sistema de reconocimiento de matrículas.

Por último, Firebase pone al alcance una funcionalidad denomina 'Firebase Hosting' que permite hacer **accesible nuestra aplicación a cualquier dispositivo del mundo**. Este paso se consigue publicando la aplicación en su sistema de hosting, y el propio firebase se encarga de distribuirla por diversos servidores Content Delivery Network (CDN) repartidos por todo el mundo. Por otra parte, también se encarga de aprovisionar y configurar un certificado Secure Sockets Layer (SSL) para la aplicación publicada, lo cual asegura que la transferencia de datos entre el cliente y el servidor esté cifrada.

5.3. Integración de los subsistemas

Una vez desarrollados ambos subsistemas por separado, es momento de integrarlos y hacerlos funcionar bajo el mismo sistema.

La **pieza angular del sistema** y la que hace de enlace entre ambos subsistemas **es la base de datos**. A la cual se accede para ingresar y consultar la información requerida en cada instante.

En el **primer paso** de la figura 5.18, el usuario interactúa con la aplicación web para crear una cuenta, ingresar en ella, agregar vehículos y solicitar pertenecer a alguna organización o alquilar un garaje. Para realizar todas estas acciones no es necesario integrar ni desarrollar nada más que lo ya

realizado durante la fase de desarrollo de la aplicación web.

En el **segundo paso** de la figura 5.18, el mismo usuario acude a la puerta del garaje al que desee acceder. El sistema de reconocimiento de matrículas es el encargado de obtener la matricula del vehículo presente. Se necesita que el sistema de reconocimiento de matrículas compruebe si dicha matrícula está asociada al garaje. Para ello, integramos el módulo SDK de firebase, tal y como se puede observar en el apéndice D.3, en este subsistema para realizar consultas contra la base de datos.

El sistema de reconocimiento de matrículas realiza dos llamadas a la base de datos: en la primera se consulta si la matrícula en texto plano pertenece al garaje, obteniendo una respuesta rápida en el **tercer paso** de la figura 5.18. En la segunda llamada envía la imagen realizada previamente al vehículo para que esta se muestre en el panel del administrador y tenga un histórico de los vehículos.

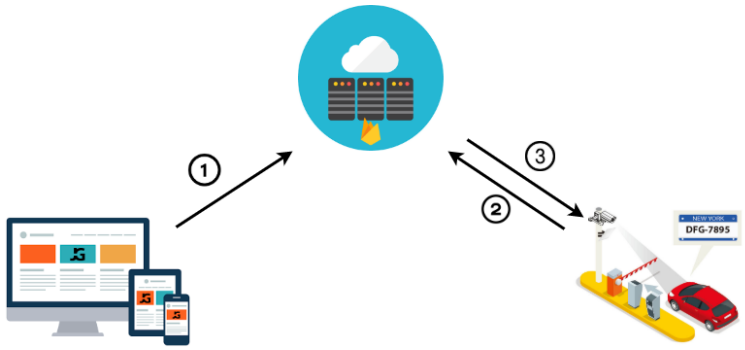

Figura 5.18: Integración de los subsistemas.

6

Pruebas y resultados

Las pruebas son un conjunto de actividades que **se realizan para comprobar el correcto funcionamiento del software desarrollado y que se cumplan con los requisitos establecidos durante la fase de análisis**. Son actividades que deben llevarse a cabo durante todo el ciclo de vida del producto. En este proyecto, se han realizado pruebas durante el desarrollo y puesta en producción del software.

Para cubrir todos los aspectos establecidos en la fase de análisis, se van a realizar pruebas funcionales y no funcionales.

6.1. Pruebas funcionales

Son pruebas específicas y exhaustivas que tienen como propósito **validar que el producto hace lo especificado en los requisitos funcionales**. Esta fase sólo incluye pruebas de caja negra, es decir, partiendo de una entrada se debe obtener la salida esperada, sin tener en cuenta el funcionamiento interno.

6.1.1. Pruebas unitarias

Tienen como propósito aislar partes del programa y comprobar que su funcionamiento es correcto. Es un tipo de prueba que se ha usado durante el desarrollo de cada una de las funcionalidades planificadas para el desarrollo del proyecto. La principal ventaja que aporta este tipo de prueba es que los errores están acotados a dicha funcionalidad y son más fáciles de localizar.

6.1.2. Pruebas de integración

Tiene como propósito que todos los elementos que conforman el sistema funcionan correctamente probándolos en grupo, evitando así posibles incompatibilidades entre ellos.
Es un tipo de prueba que realizamos tras finalizar el desarrollo de una funcionalidad y comprobar que su funcionamiento en un entorno aislado es el esperado, pues tras la integración de la nueva

funcionalidad debemos comprobar si esta no rompe el estado global de la aplicación.

6.2. Pruebas no funcionales

Son un tipo de pruebas que no **se centran en el estudio de** la funcionalidad del producto, sino en atributos del mismo como **la usabilidad, compatibilidad y rendimiento, entre otros**.

6.2.1. Prueba de usabilidad

Es un tipo de prueba que sirve para evaluar un producto mediante pruebas con usuarios finales. Consiste en seleccionar a un grupo de usuarios y solicitar que realicen tareas para el cual fue diseñado el producto, de esta manera se obtiene interacción del usuario y se observa si encuentran dificultades durante su realización. Este tipo de prueba ha sido utilizado durante el desarrollo de la aplicación web para comprobar si resultaba cómoda y fácil de usar.

6.2.2. Pruebas de compatibilidad

Sirven para verificar que el producto funcione de manera correcta en aquellos entornos donde debería hacerlo. Se ha realizado esta prueba sobre la aplicación web, de esta manera se ha podido probar su correcto funcionamiento en los navegadores con más cuota de mercado.

6.2.3. Pruebas de rendimiento

Usadas en el sistema de reconocimiento de matrículas para comprobar el rendimiento del mismo. La principal métrica observada es el tiempo empleado para ello, obteniéndose un resultado dentro de lo estipulado en la fase de análisis.

7

Conclusión y Trabajo futuro.

7.1. Conclusión

Tras la realización de este TFG **se han conseguido los objetivos estipulados**: el desarrollo de un sistema conformado tanto por una red social donde los usuarios y administradores pueden gestionar sus garajes, como por un software para el reconocimiento de matriculas. Por dicha razón, se considera que la realización del proyecto es satisfactoria.

Se ha adquirido experiencia sobre cómo funciona y qué fases hay que realizar desde que surge una idea de negocio hasta que se obtiene el producto final, si es que llega.

Se ha aprendido la **importancia qué tienen algunas de estas fases puesto que determinan la propia viabilidad del proyecto**. Como por ejemplo, la investigación realizada durante la fase de análisis del proyecto sobre el conjunto de tecnologías y herramientas que se van a usar en él.

El uso de Firebase, con su arquitectura BaaS , permitió que el desarrollo no fuera inviable pues liberaba tiempo de desarrollo. Por otro lado, la existencia de librerías como OpenCV nos ha permitido realizar el tratamiento y procesado de imágenes de una manera muy sencilla. Además de haber acertado con las tecnologías y herramientas, también ha sido satisfactorio el hecho de haberlas aprendido.

Por otro parte, también **se han aplicado conocimientos que han sido adquiridos en varias asignaturas del Grado de Ingeniería Informática**. En concreto, se han utilizado conocimientos sobre diseño de proyectos y obtención de los requisitos aprendidos de las asignaturas *Ingeniería del Software* y *Proyecto de Análisis y Diseño de Software*. Conocimientos para la adquisición, procesamiento, análisis y comprensión de objetos adquiridos en las asignaturas *Inteligencia Artificial* y *Fundamentos de aprendizaje Automático*. Conocimientos sobre desarrollo web de la asignatura *Sistemas Informáticos*.

Durante el transcurso del proyecto se ha convivido con etapas arduas que propiciaban pensamientos negativos acerca de la magnitud del proyecto y sobre su realización. A pesar de las dificultades, me siento muy satisfecho con el resultado final del proyecto pues he sido capaz de abordar y resolver todas ellas.

7.2. Trabajo futuro

Aunque los objetivos estipulados han sido satisfechos en el desarrollo de este TFG , a continuación se exponen **posibles mejoras que se pueden aplicar** sobre el mismo como trabajo futuro:

En cuanto a la aplicación web, se podría trabajar en:

Realizar **mejoras del proceso de alquiler** de plazas de garajes pues en las pruebas de usabilidad realizadas se comprobó cierta dificultad para realizar dicha tarea.

También sería interesante añadir una **pasarela de pagos** para hacer efectivos los pagos de los alquileres entre los distintos usuarios.

La implementación de un **sistema de mensajería** que permitiera a usuarios y administradores ponerse en contacto.

Habilitar el multidioma en el sistema, permitiendo al usuario escoger el idioma de la aplicación. Por rapidez solo de desarrollo con un único idioma, el inglés.

En cuanto al sistema de reconocimiento de matriculas:

La **integración del software de reconocimiento de matrículas en un dispositivo portable, de bajo consumo y la potencia necesaria para el procesamiento rápido de las matrículas**. Podría realizarse con la Raspberry Pi 3, se trata de un miniordenador al que se le pueden incorporar módulos para su personalización. Para este proyecto sería necesario un módulo de cámara con visión nocturna y un sensor de proximidad para que la cámara realice las instantáneas.

La mejora de la técnica de reconocimiento de matriculas y el estudio de nuevas que **eviten la usurpación de identidades** mediante la falsificación de matrículas. Puesto que la única validación para acceder a cualquier garaje es la matrícula del vehículo.

Bibliografía

[1] *Técnicas y algoritmos básicos de visión artificial*. [Accedido: Abril 2020]. Disponible en: Researchgate.

[2] *Image Processing in OpenCV*. [Accedido: Marzo 2020]. Disponible en: OpenCV docs.

[3] *O'Reilly: learning OpenCV*. [Accedido: Marzo 2020]. Disponible en: O'Reilly book.

[4] *World Wide Web*. [Accedido: Mayo 2020]. Disponible en: Wikipedia.

[5] *Static web page*. [Accedido: Mayo 2020]. Disponible en: Wikipedia.

[6] *Escuela Politécnica Superior UAM*. [Accedido: Mayo 2020]. Disponible en: UAM website.

[7] *Amazon*. [Accedido: Mayo 2020]. Disponible en: Amazon Website.

[8] *Dynamic web page*. [Accedido: Mayo 2020]. Disponible en: Wikipedia.

[9] *Single-page application*. [Accedido: Abril 2020]. Disponible en: Wikipedia.

[10] *Image processing in Python with Scikit-image*. [Accedido: Marzo 2020]. Disponible en: Scikit-image website.

[11] *Computer Vision in Python with Mahotas*. [Accedido: Marzo 2020]. Disponible en: Mahotas website.

[12] *Image processing and computer vision with Opencv*. [Accedido: Marzo 2020]. Disponible en: Opencv website.

[13] *Dataset de vehículos para el renocomiento de matrículas*. [Accedido: Marzo 2020]. Disponible en: GitHub.

[14] *The Chars74K dataset - Character Recognition in Natural Images*. 2012. [Accedido: Marzo 2020]. Disponible en: Website.

[15] *Vuetify: Material Design Component Framework*. [Accedido: Abril 2020]. Disponible en: Vuetify website.

[16] *Vuex: state management pattern*. [Accedido: Abril 2020]. Disponible en: Vuex website.

[17] *Vue-router: building Single Page Applications*. [Accedido: Abril 2020]. Disponible en: Vue-router website.

[18] *Firebase: a mobile and web application development platform*. [Accedido: Abril 2020]. Disponible en: Firebase website.

[19] *Google Maps Platform*. [Accedido: Abril 2020]. Disponible en: Google Api Website.

[20] *Maps and location for developers*. [Accedido: Abril 2020]. Disponible en: Mapbox website.

[21] *Automatic Number Plate Recognition System (ANPR): A Survey*. 2013. [Accedido: Marzo 2020]. Disponible en: Researchgate.

[22] *Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study*. 2012. [Accedido: Abril 2020]. Disponible en: Researchgate.

[23] *Repositorio que contiene todo el código desarrollado en este TFG.* [Accedido: Mayo 2020]. Disponible en: GitHub repository.

[24] *Viola–Jones object detection framework*. [Accedido: Marzo 2020]. Disponible en: Wikipedia.

[25] *OpenCV tutoriasl - Cascade Classifier*. [Accedido: Marzo 2020]. Disponible en: OpenCV docs.

[26] *Clasificador haar cascade entrenado con matrículas rusas*. 2014. [Accedido: Marzo 2020]. Disponible en: GitHub.

[27] *Tipos de matrículas rusas*. [Accedido: Marzo 2020]. Disponible en: Website.

[28] *Vehicle registration plates of Europe*. [Accedido: Marzo 2020]. Disponible en: Wikipedia.

[29] *Vue framework de JavaScript*. [Accedido: Abril 2020]. Disponible en: Vue website.

[30] *API documentation of Mapbox GL JS*. [Accedido: Abril 2020]. Disponible en: Mapbox API.

[31] *Haar Cascade tutorial*. [Accedido: Marzo 2020]. Disponible en: OpenCV docs.

[32] *Get Started with Firebase Authentication on Websites*. [Accedido: Mayo 2020]. Disponible en: Firebase docs.

[33] *Create custom email action handlers*. [Accedido: Abril 2020]. Disponible en: Firebase docs.

[34] *Get started with Firebase Hosting*. [Accedido: Mayo 2020]. Disponible en: Firebase docs.

Definiciones

dataset Es un conjunto de datos que forman una colección.

framework Es un conjunto estandarizado de prácticas y conceptos que sirve como referencia para resolver problemas de índole similar.

función hash Son funciones matemáticas que a partir de una misma cadena de entrada producen una misma cadena de salida. Entre sus usos destaca su utilización en el enmascaramiento de contraseñas, permitiendo que no sean guardas en las bases de datos en texto plano. No confundir con funciones de cifrado.

opensource El software opensource hacen referencia a proyectos con código accesible al público: todos pueden ver, modificar y distribuir el código de la forma que consideren conveniente.

salt Cadenas aleatorias que se añaden a la contraseña antes de ser procesada por la función hash. De esta manera se obtienen valores hash con menor probabilidad de estar precalculados en diccionarios arcoíris.

Acrónimos

BaaS Backend as a Service.

CDN Content Delivery Network.

CLI Command Line Interface.

HOG Histogram of Oriented Gradients.

HTML HyperText Markup Language.

JSON JavaScript Object Notation.

KNN K-Nearest Neighbors.

MVVM Modelo, Vista, Vista-Modelo.

NoSQL Not only SQL.

OCR Optical Character Recognition.

ppi pixeles por pulgada.

RGB Red Green Blue.

SDK Software Development Kit.

SPA Single Page Aplication.

SSL Secure Sockets Layer.

SURF Speeded-Up Robust Features.

TFG Trabajo de Fin de Grado.

WWW World Wide Web.

Apéndices

A

Crear clasificador Haar Cascade con OpenCV

Este apéndice sirve como guía para crear un clasificador propio Haar Cascade [31]. Los pasos que se van a tratar son: obtención de los datos de entrenamiento, entrenamiento del modelo y su posterior puesta a prueba.

El paso más importante es la **obtención de un dataset** con imágenes de calidad y en abundancia, pues para entrenar el clasificador se necesita una gran cantidad de imágenes positivas e imágenes negativas. Ambos dataset se han de crear manualmente, con el cuidado de que las imágenes negativas no contengan el objeto a clasificar. Una vez obtenidos, es necesario ubicar y etiquetar manualmente las imágenes positivas. Para ello, se hace uso del siguiente comando:

```
$ opencv_annotation --annotations=objects-detected.txt
  --images=positives
```

El comando anterior abre una ventana que muestra una a una las imágenes positivas ubicadas en el directorio 'positives'. El objetivo de este proceso es definir el modelo que debería ser capaz de encontrar el objeto a detectar. Para ello, en cada una de las imágenes se debe ir marcando manualmente dónde está ubicado dicho objeto. Al terminar con todo el dataset de imágenes positivas, se genera un archivo denominado 'objects-detected.txt' que presenta el siguiente formato:

```
positives/P1010001.jpg 1 217 331 186 38
positives/P1010002.jpg 1 205 260 217 55
positives/P1010003.jpg 1 197 230 238 61
...
```

Cada fila corresponde a una imagen. La primera columna representa el nombre de la imagen que ocupa en el directorio 'positives'. La segunda es el número de objetos que fueron ubicados en esa imagen, en este caso, se buscaron imágenes que solo contuvieran una matrícula. Por último, se muestra la ubicación del objeto. La ventana que se utiliza es de tipo rectangular, por lo que con la definición de un punto para la esquina superior izquierda (los dos siguientes valores) y otro punto para la esquina inferior izquierda (los dos últimos valores) es suficiente.

El siguiente paso es **entrenar al clasificador**. Se basará en el estudio de las imágenes positivas y negativas tratadas anteriormente. Esta fase se realiza por medio del siguiente comando:

```
$ opencv_traincascade -data classifiers/own/ -vec object_detected.vec
-bg negatives.txt -numPos 400 -numNeg 217 -numStages 15 -w 48 -h 24
-featureTypes HAAR -minHitRate 0.95
```

Con el parámetro '-data' se define la ubicación que ocupará el clasificador creado. El parámetro '-vec' contiene el vector de imágenes positivas creadas con anterioridad. El parámetro '-bg' contiene un listado con la ubicación en el proyecto de cada una de las imágenes negativas. El parámetro '-numPos' representa el número de imágenes positivas usadas por cada iteración del clasificador. El parámetro '-numNeg' igual que el anterior pero sobre imágenes negativas. El parámetro '-numStages' representa el número de iteraciones o etapas que será entrenado el clasificador. '-featureTypes' representa el tipo de características que se aplicarán durante la fase de entrenamiento, en este caso, son características de tipo HAAR. Y por último, '-minHitRate' representa la mínima puntuación de éxito deseada en cada iteración.

Una vez que el clasificador ha sido entrenado solo queda probarlo, para ello se crea el siguiente código A.1. En él se van recorriendo todas las imágenes ubicadas en el directorio '/Dataset/Cars/' para comprobar el resultado del clasificador.

Código A.1: Código necesario para ejecutar el clasificador previamente entrenado.

```
1  if __name__ == '__main__':
2
3      ## Load the classifier.
4      # plate_cascade = cv.CascadeClassifier('./classifiers/haarcascade_russian_plate_number.xml')
5      plate_cascade = cv.CascadeClassifier('./classifiers/own/cascade.xml')
6
7      directoryPath = '../../../Dataset/Cars/'
8      for imagePath in os.listdir(directoryPath):
9          img = cv.imread(directoryPath + imagePath)
10
11         imageGray, ret, thresh, contours, hierarchy = processImage(img)
12
13         ## Using classifier.
14         plates = plate_cascade.detectMultiScale(imageGray, 1.01, 7)
15
16         ## Painting a rectangle on the license plates found.
17         for (x,y,w,h) in plates:
18             img = cv.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
19
20         ## Showing this image with rectangle.
21         showImages({'img': img})
```

B

Diagramas

B.1. Casos de uso

El diagrama de casos de uso **sirve para describir las actividades que se han de realizar para llevar a cabo algún proceso**. En la figura B.1 se pueden distinguir dos tipos de actores:

Usuario personal:

Puede añadir vehículos a su cuenta y su posterior actualización o eliminación.

Gestión de las plazas de garajes de las organizaciones a las que pertenezca y controlar el alquiler de las mismas.

Solicitar alquileres de plazas de garajes de otros usuarios personales mediante su búsqueda en un mapa interactivo.

Búsqueda de organizaciones por nombre o en un mapa interactivo para su posteriormente enviar solicitud de ingreso.

El usuario administrador:

Validación de las solicitudes recibidas por los usuarios personales y su posterior actualización de datos y eliminación de usuarios.

Acceso al panel principal donde puede gestionar métricas como la ocupación del garaje o un histórico de vehículos.

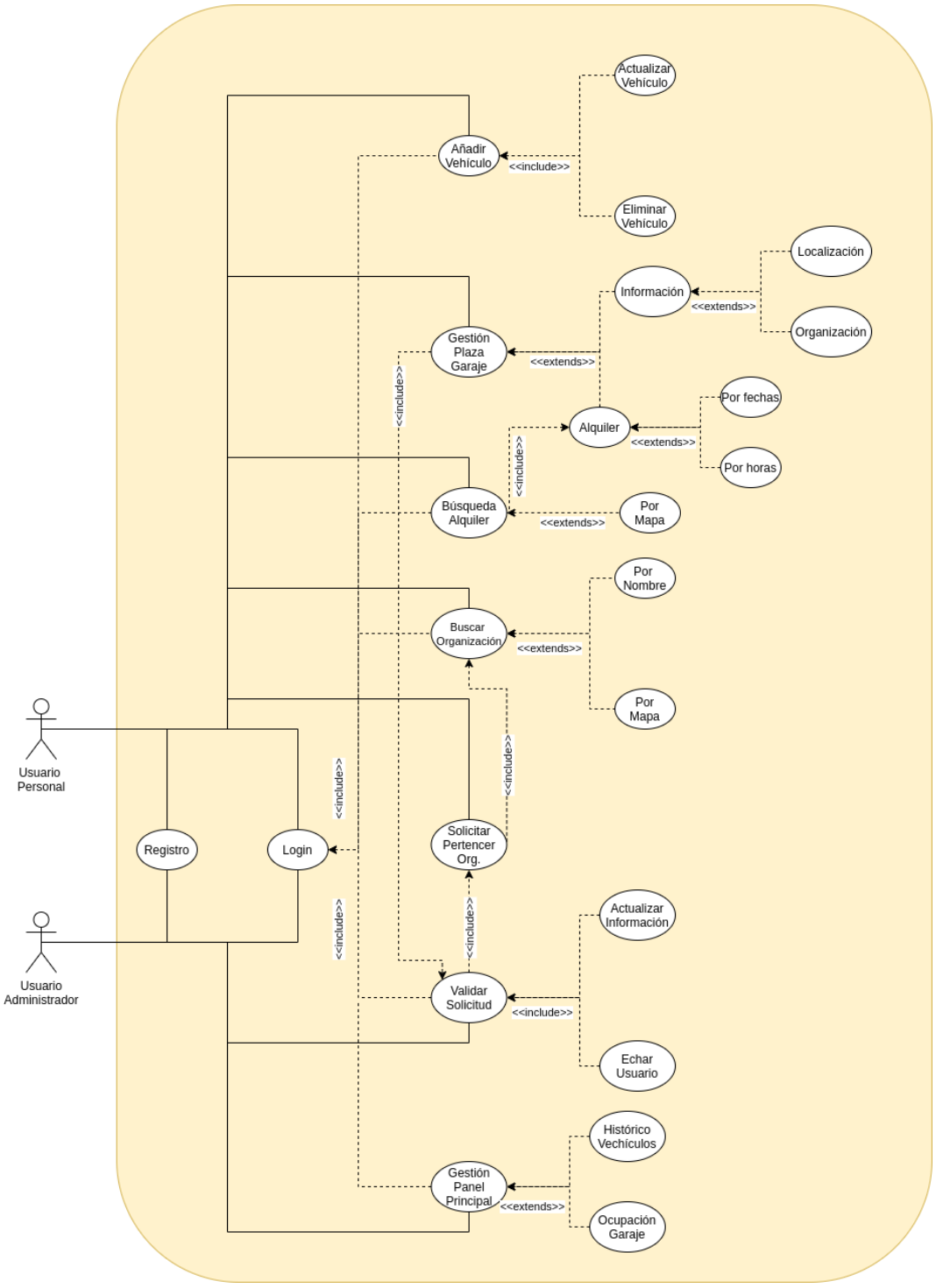

Figura B.1: Casos de uso de la aplicación web.

B.2. Arquitectura de la base de datos

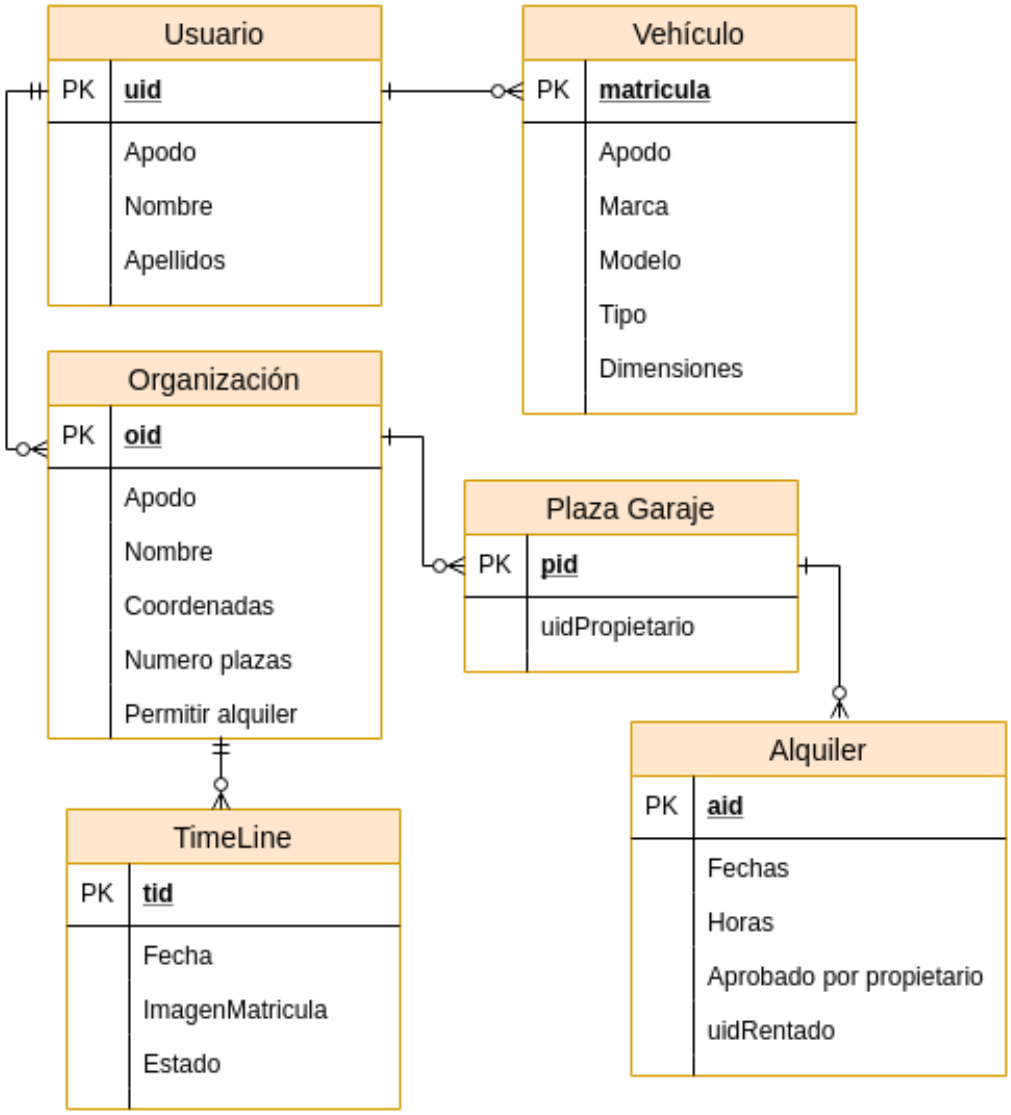

Figura B.2: Arquitectura de la base de datos NoSQL de tipo documental.

C

Interfaz y flujo de la aplicación web

En este anexo **se muestra el diseño final de las distintas vistas que componen la aplicación web**. Se distinguen tres tipos de vistas: vistas comunes, del usuario administrador y del usuario personal.

C.1. Vistas comunes

Estas vistas corresponden a las partes de la aplicación donde cualquier tipo de usuario tiene acceso, es decir, aquellas donde el usuario aún no se ha autenticado [32]: login y registro en la aplicación.

C.1.1. Login

Como se observa en la figura C.1 se muestra la pantalla de login para acceder a la aplicación. No hace falta indicar con qué tipo de usuario se quiere acceder pues en el registro ya se escoge qué tipo de usuario está vinculado a dicho usuario.

C.1.2. Registro

Se parte de una vista (figura C.2) que comparten ambos tipos de usuarios. En esta vista es necesario introducir un correo electrónico, una contraseña, su repetición y el tipo de usuario que se desee. En función de que tipo de usuario se escoja se redirigirá al registro específico de ese tipo de usuario.

El registro de usuarios personales se puede observar en la figura C.3. El usuario deberá introducir un apodo, su nombre, apellidos y el país donde se encuentra.

Mientras que en **el registro de usuarios administradores** hay que introducir el nombre de la organización, el número de plazas de garaje y la ubicación de la puerta del garaje en el mapa interactivo. Esto se puede observar en las figura C.4.

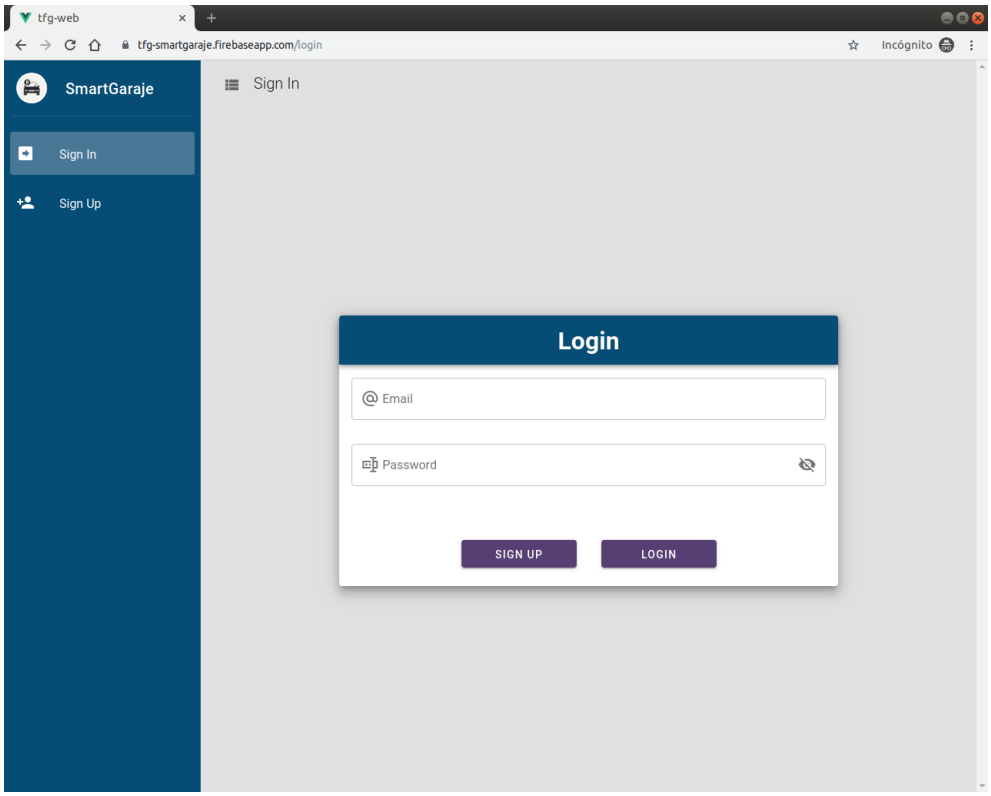

Figura C.1: Pantalla de login.

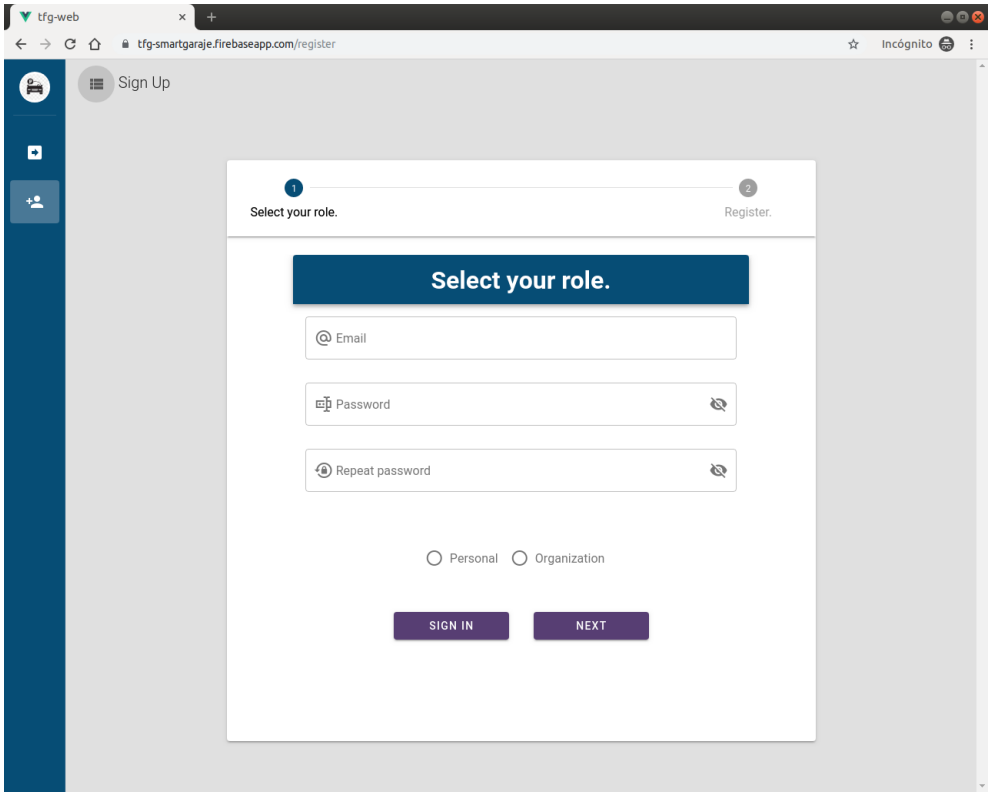

Figura C.2: Pantalla de registro común.

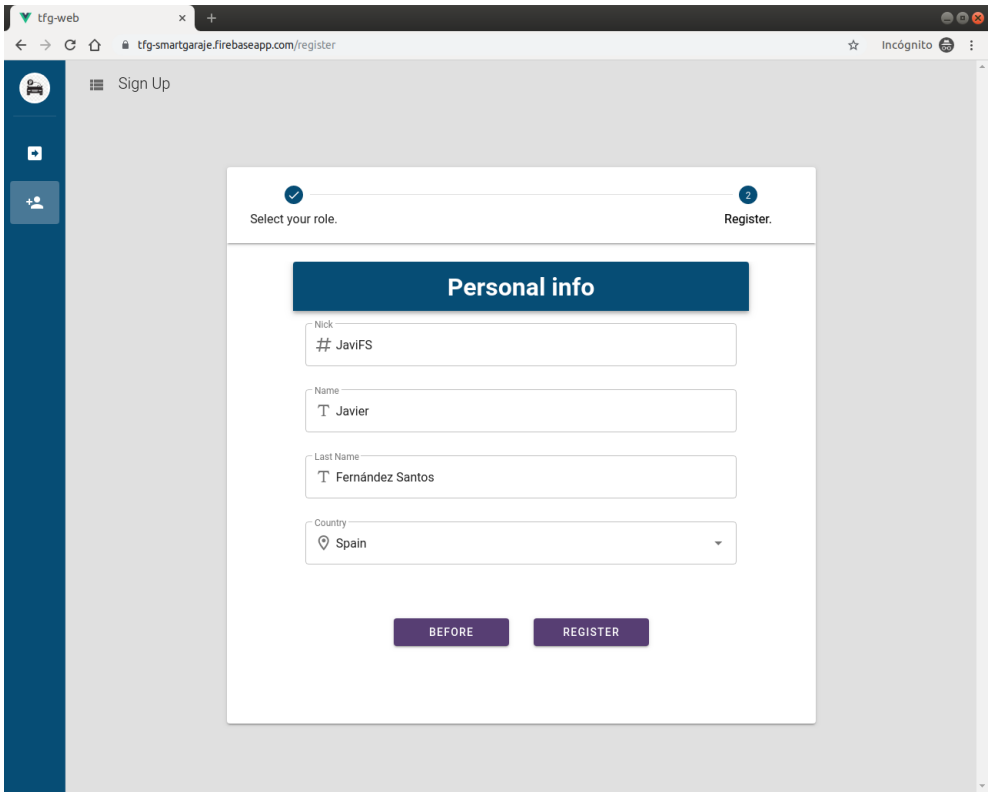

Figura C.3: Pantalla de registro personal con datos.

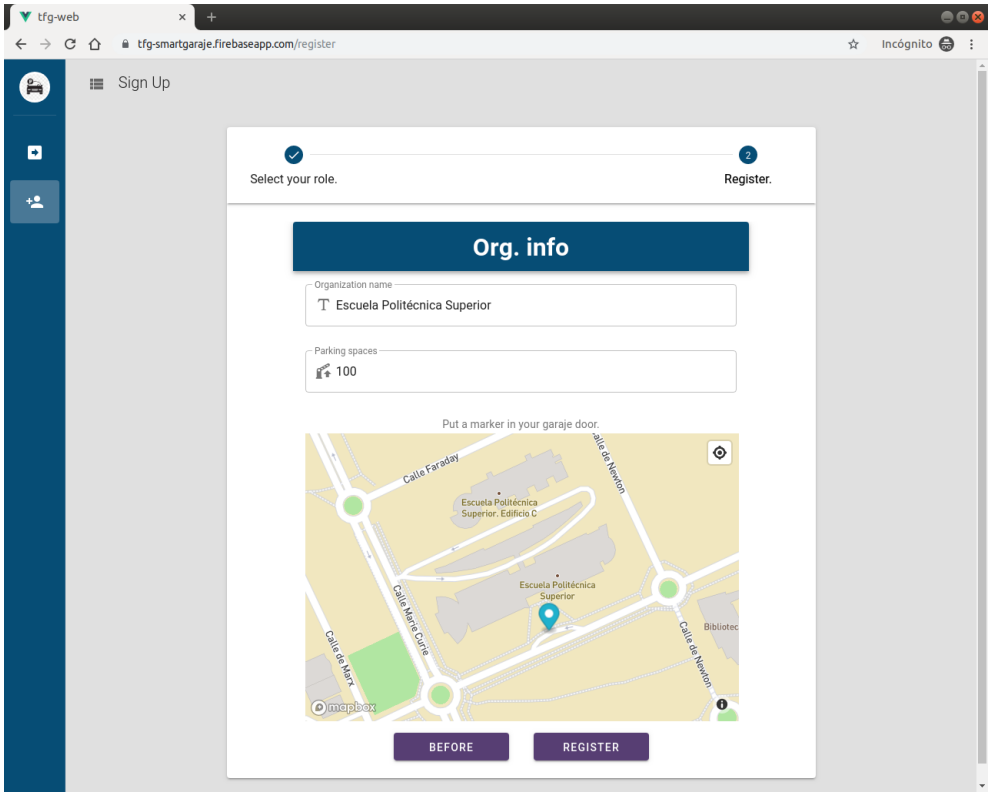

Figura C.4: Pantalla de registro administrador con datos.

Por último, se dispone de un mecanismo para la verificación del usuario mediante **validación por correo electrónico** [33], independientemente del usuario escogido para el registro.

Una vez registrado, el usuario es redirigido a la vista correspondiente a la figura C.5. Hasta que no confirme el registro por medio del mensaje recibido en su correo electrónico no podrá interactuar con la aplicación.

El mensaje que el usuario recibe en la bandeja de entrada de su correo electrónico es como se muestra en la figura C.6. Tras pinchar sobre el enlace recibido en el mensaje, el usuario activará su cuenta y ya podrá acceder a toda la funcionalidad de la aplicación, tal y como vemos en la figura C.7

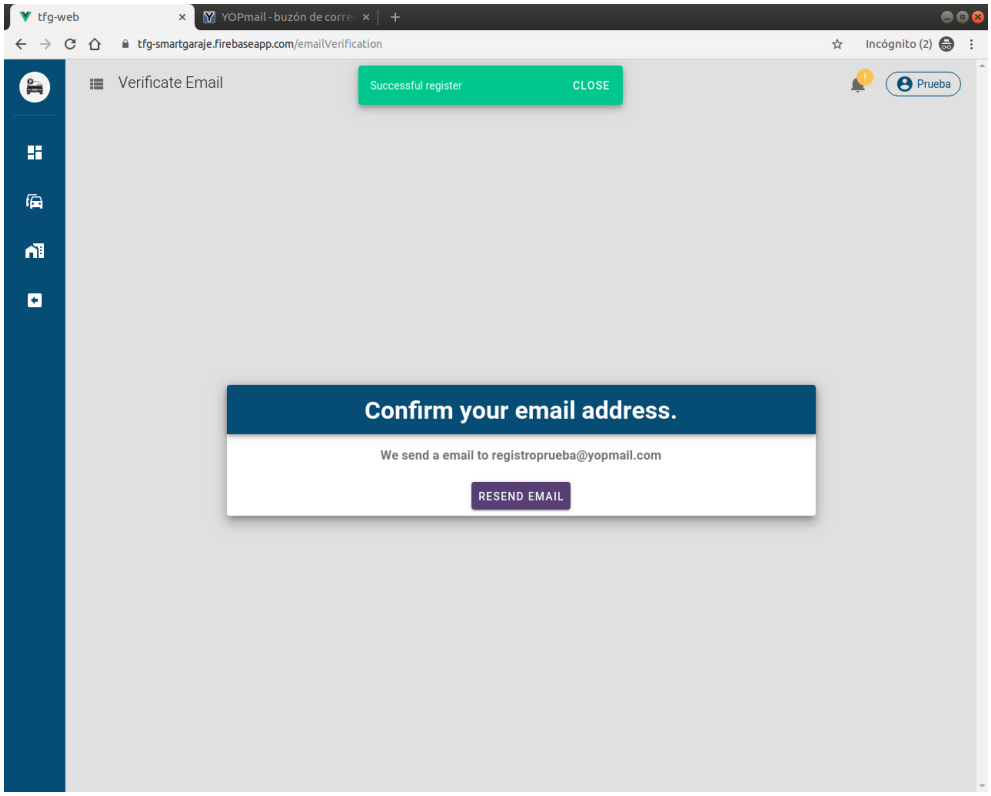

Figura C.5: Cuenta registrada pero sin verificar.

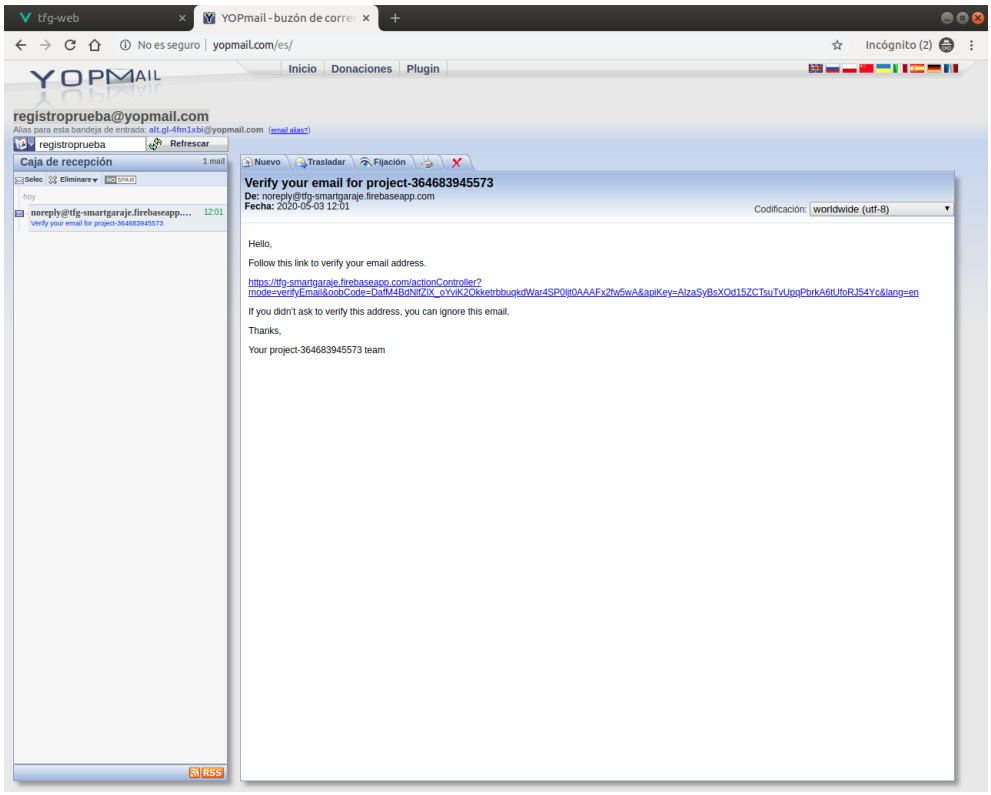

Figura C.6: Verificación por correcto electrónico.

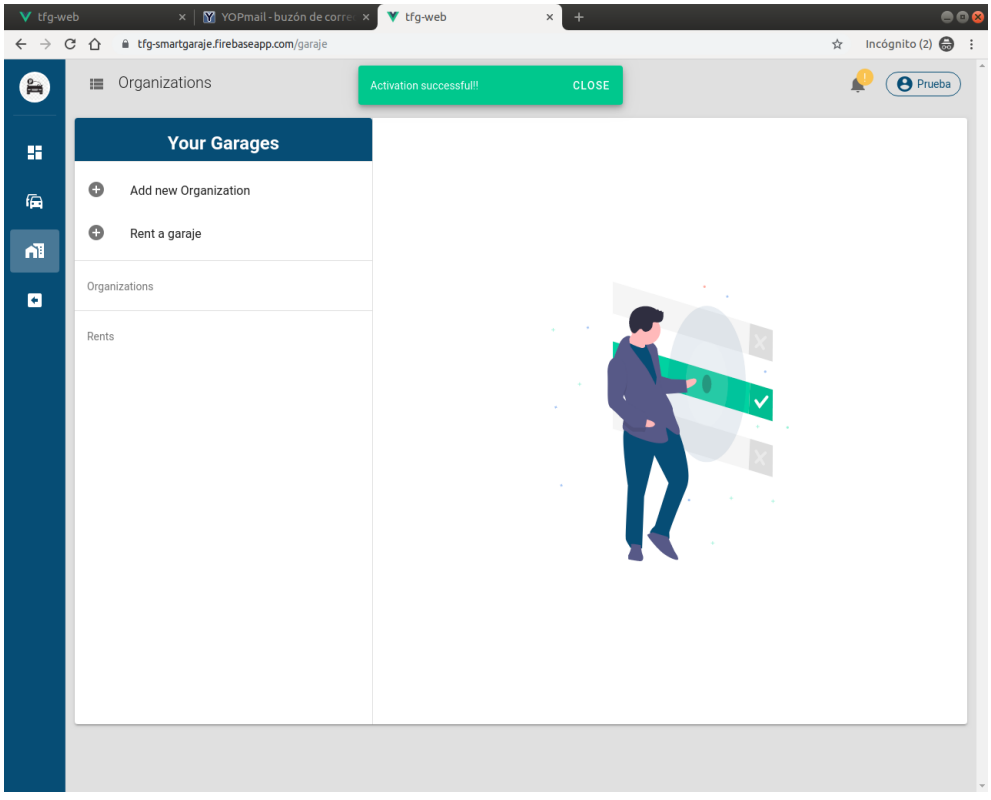

Figura C.7: Cuenta verificada.

C.2. Vistas del usuario administrador

El usuario administrador solo dispone de dos tipos de páginas: un panel principal con el que observar el estado del garaje y un panel de administrador para gestionar a los usuarios.

En **el panel principal** (figura C.8) se pueden observar dos métricas: el estado de ocupación del garaje y un histórico de los vehículos que han pasado por él.

En **el panel de administración de usuarios** el administrador dispone de dos ventanas. La primera (figura C.9) le sirve para gestionar las solicitudes que recibe de usuarios personales para formar parte de la organización. Dispone de dos tipos de acciones: denegar o aceptar la solicitud. En caso de aceptarla, tendrá que indicar el número de plazas que le corresponden a dicho usuario y a partir de ese instante el usuario ya podrá acceder a este garaje con los vehículos registrados en su cuenta.
La segunda ventana (figura C.10) presenta una lista de los usuarios que ya pertenecen a la organización. En esta lista el administrador puede modificar la información de los usuarios o eliminarlos de la organización.

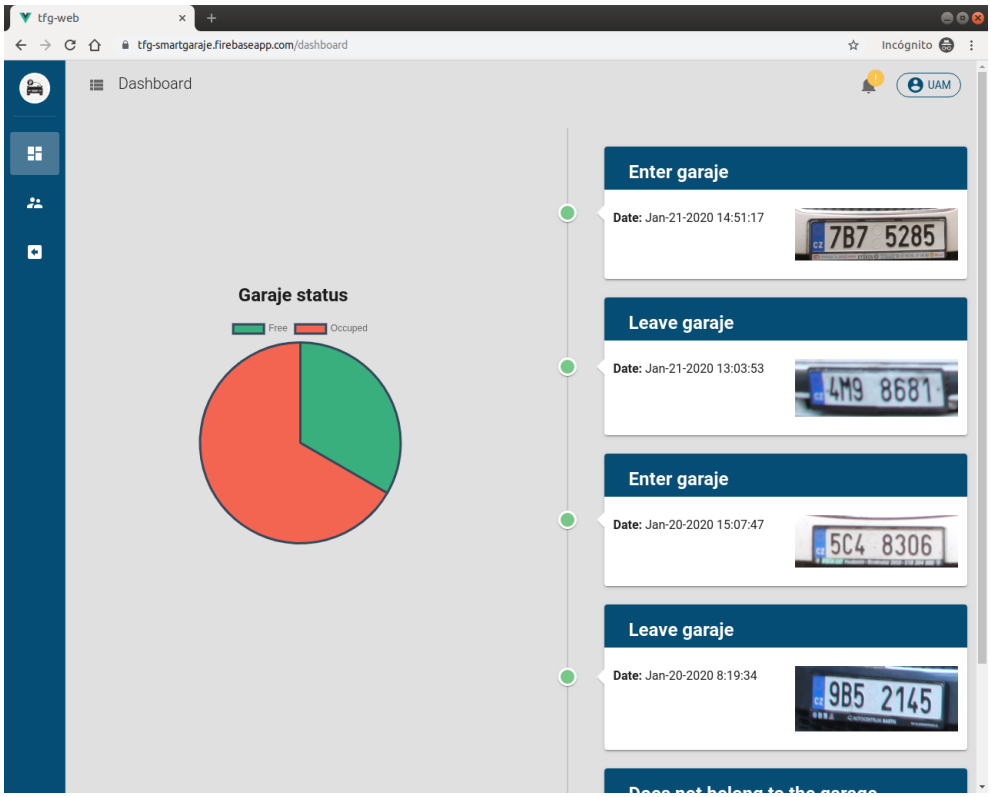

Figura C.8: Pantalla del dashboard del administrador.

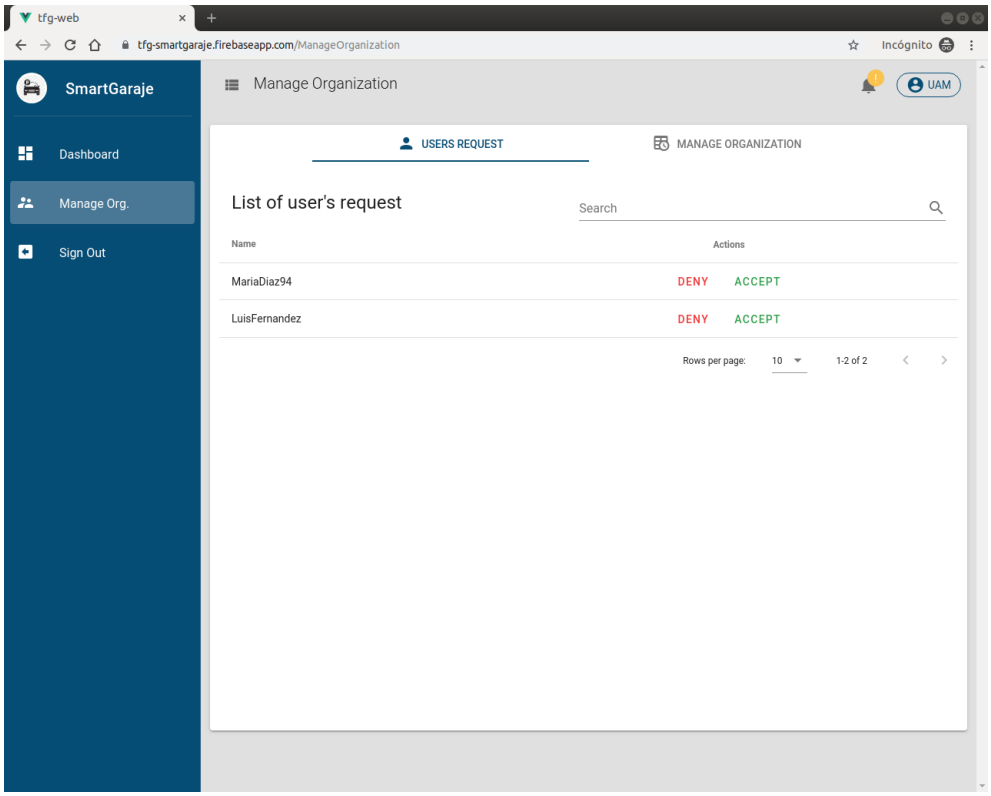

Figura C.9: Lista de solicitudes de usuarios personales para pertenecer a esta organización.

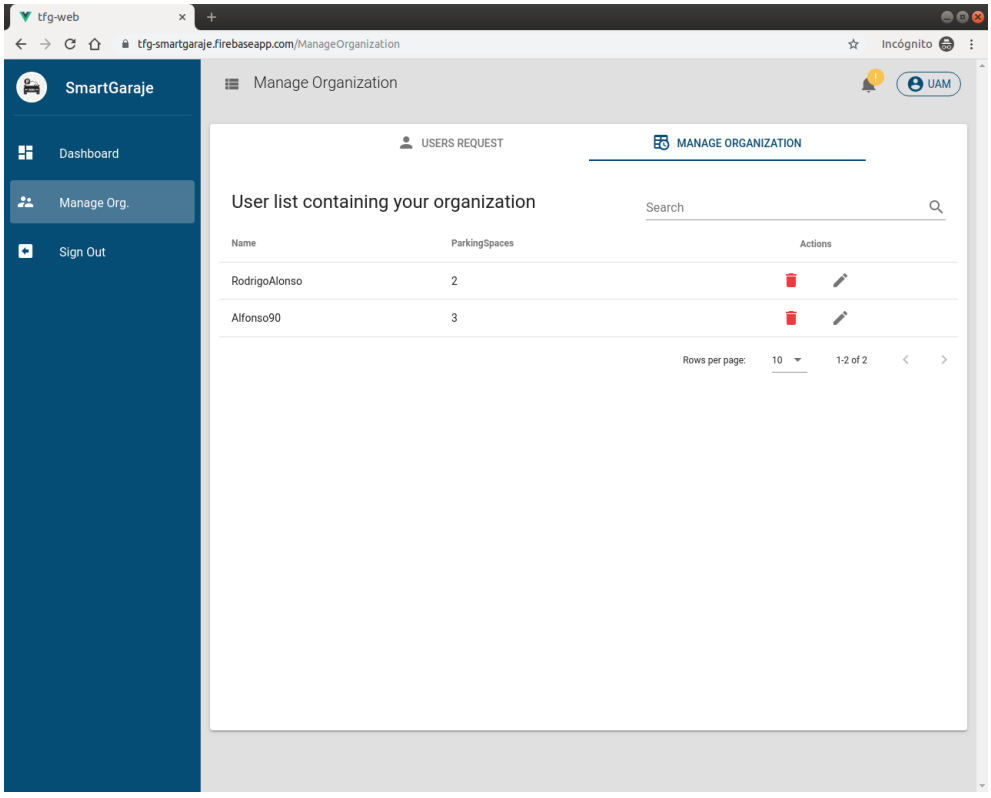

Figura C.10: Lista para gestionar los usuarios pertenecientes a la organización.

C.3. Vistas del usuario personal

Una de las primeras opciones que presenta el usuario personales en el menú de la izquierda es un dashboard. La idea era integrar resúmenes e información pero al final no se ha usado.

La siguiente opción son los **vehículos del usuario**. Aquí el usuario podrá añadir tantos vehículos como desee. Cuando el usuario se dirija a algún garaje y el sistema de reconocimiento de matrículas obtenga la matrícula, comprobará que algún usuario perteneciente a ese garaje tenga un vehículo con dicha matrícula.

Por defecto, cuando el usuario no disponga de ningún vehículo se mostrará la vista de la figura C.11.

El usuario puede agregar un nuevo vehículo presionando sobre el botón flotante ubicado en la esquina inferior derecha. Al presionar dicho botón se abre un modal (figura C.12) que debe rellenar con las características del vehículo.

Una vez guardadas las características del vehículo, este se añadirá a la cuenta. Se puede ver en la figura C.13 la apariencia que tiene. Por defecto la única información que se muestra del vehículo es el apodo que se le ha asignado y su matrícula. Si se desea ver más información del vehículo se debe presionar la flecha para que se despliegue dicha información. En ese mismo desplegable existe la posibilidad eliminar el vehículo de la cuenta o de modificar la información del mismo.

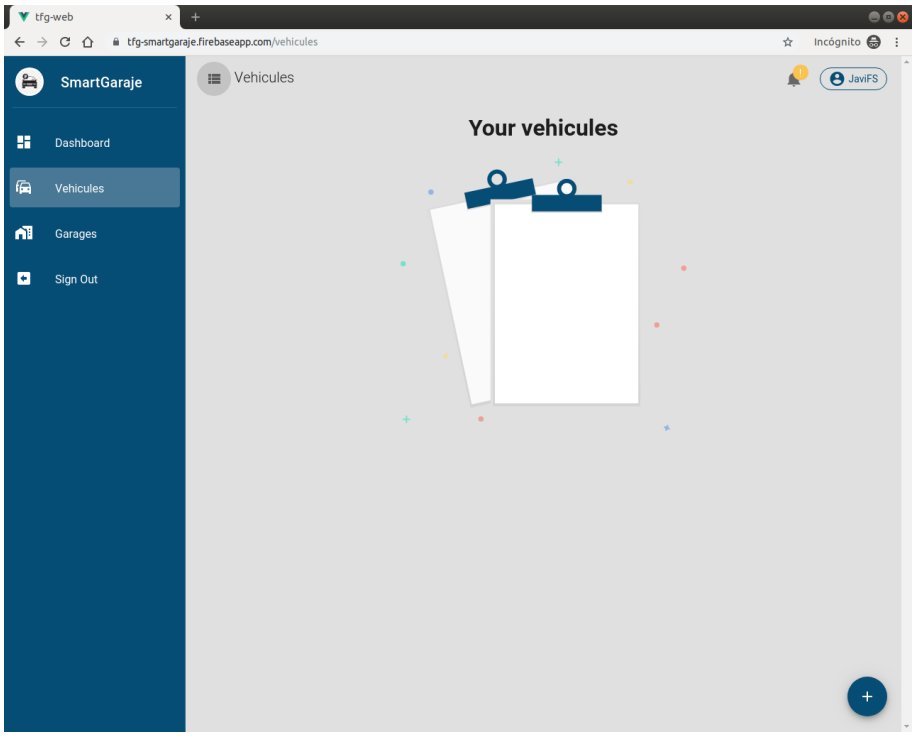

Figura C.11: El usuario no dispone de vehículos por el momento.

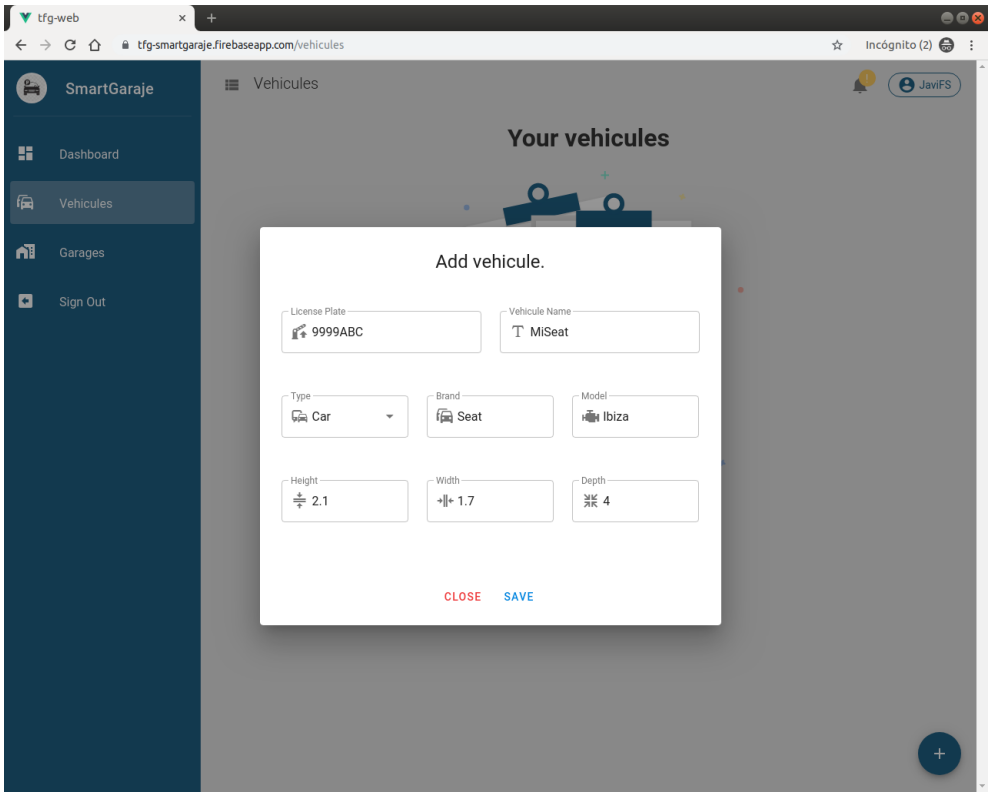

Figura C.12: Añadiendo un vehículo a la cuenta.

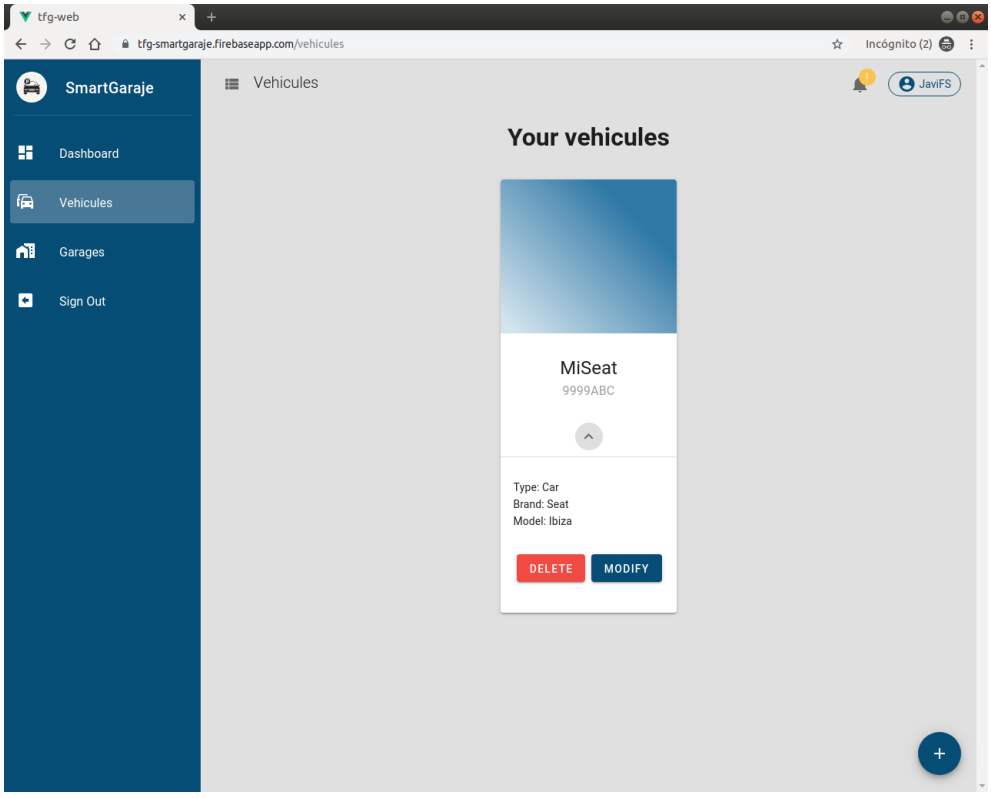

Figura C.13: El usuario tiene un vehículo.

La siguiente opción corresponde con los **garajes que tiene el usuario**.
La vista por defecto de los garajes es la que se puede observar en la figura C.14. Se distinguen dos **tipos de garajes**. Por un lado existen los llamados **garajes de organizaciones**: para poder acceder a estos garajes has de solicitar al usuario administrador de dicha organización permiso para pertenecer a la misma (por ejemplo: el garaje del trabajo, comunidad de vecinos, etc).
Por otro lado, existen **garajes rentados a otros usuarios personales**: los cuales han hecho públicos esos garajes para sacar rentabilidad (por ejemplo: alquilar la plaza de garaje cuando no se esté utilizando).

Si se quiere **solicitar acceso a una nueva organización** se debe presionar sobre el botón de añadir nueva organización en la esquina superior izquierda presenta en la figura C.14. Tras presionar sobre dicho botón se abrirá un modal que contiene una lista (figura C.15) o un mapa interactivo (figura C.16) con todas las organizaciones disponibles.
Tras solicitar acceso a cualquier organización, se agrega a la lista de organizaciones pero todavía no se puede acceder ni a sus instalaciones ni a la información de las plazas, pues todavía hace falta que el administrador gestione la solicitud. Este comportamiento se puede observar en la organización 'UAM Politécnica Parking' presente en la lista de la figura C.14.

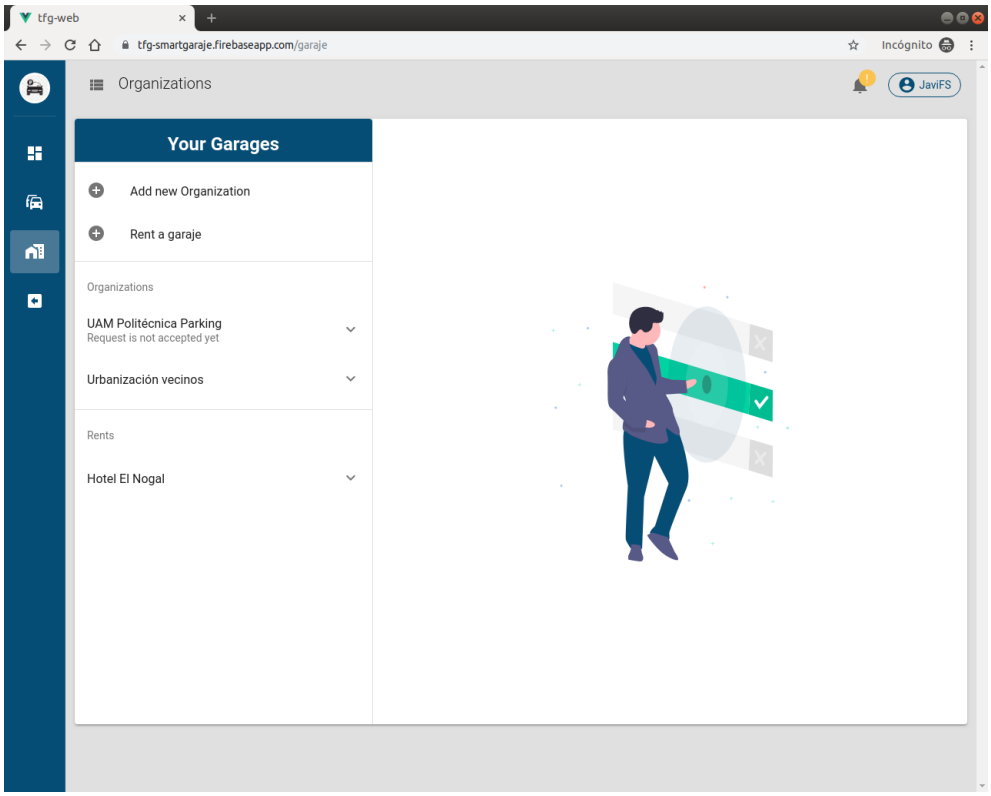

Figura C.14: Vista predeterminada del garaje.

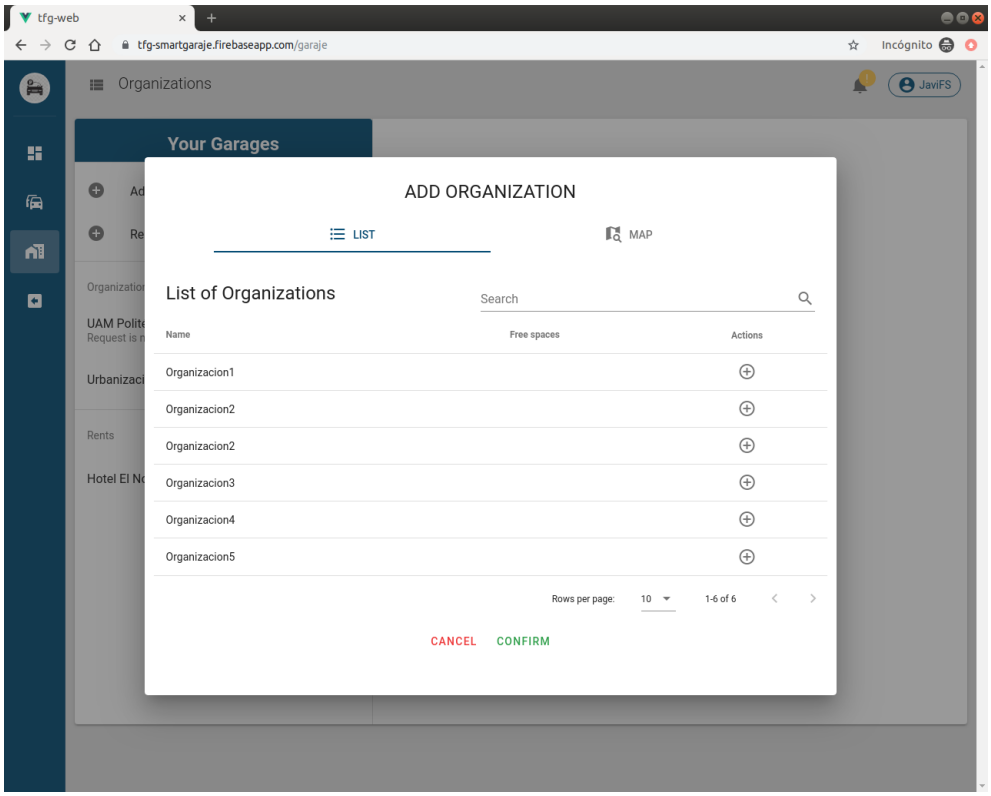

Figura C.15: Añadir organización mediante lista.

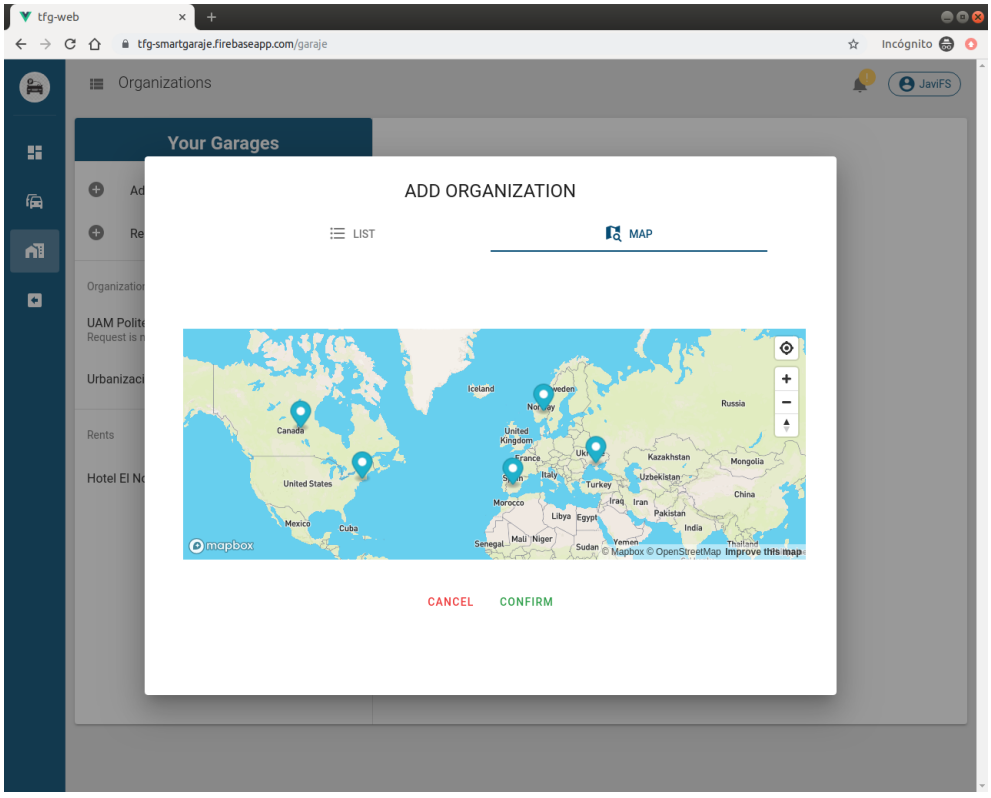

Figura C.16: Añadir organización mediante mapa interactivo.

Por otro lado, **si la solicitud ya ha sido aceptada por el administrador** de la organización se podrá acceder al garaje de la misma. En la figura C.17 se puede ver que en la organización 'Urbanización vecinos' el administrador ha asignado al usuario personal 'JaviFS' dos plazas de garaje (Parking0 y Parking1). Si se accede a la primera de ellas se puede ver su información y da la posibilidad de publicar la plaza de garaje para que otro usuario personal la pueda alquilar.

Podríamos suponer que solo se está haciendo uso de una de las dos plazas de garaje. Partiendo de este supuesto, **se podría publicar una de ellas para que otro usuario personal pagara por su uso**. Haciendo click sobre 'Rent your Garage' en la figura C.17 se mostrará la vista correspondiente con la figura C.18. En esta vista aparece un calendario, en él se debe seleccionar los días en los que la plaza esté libre, la hora a la que lo estará (C.19) y la hora que dejará de estarlo (fig. C.20). Tras publicar la plaza de garaje, esta quedará publicada y cualquier otro usuario personal podrá reservar la plaza.

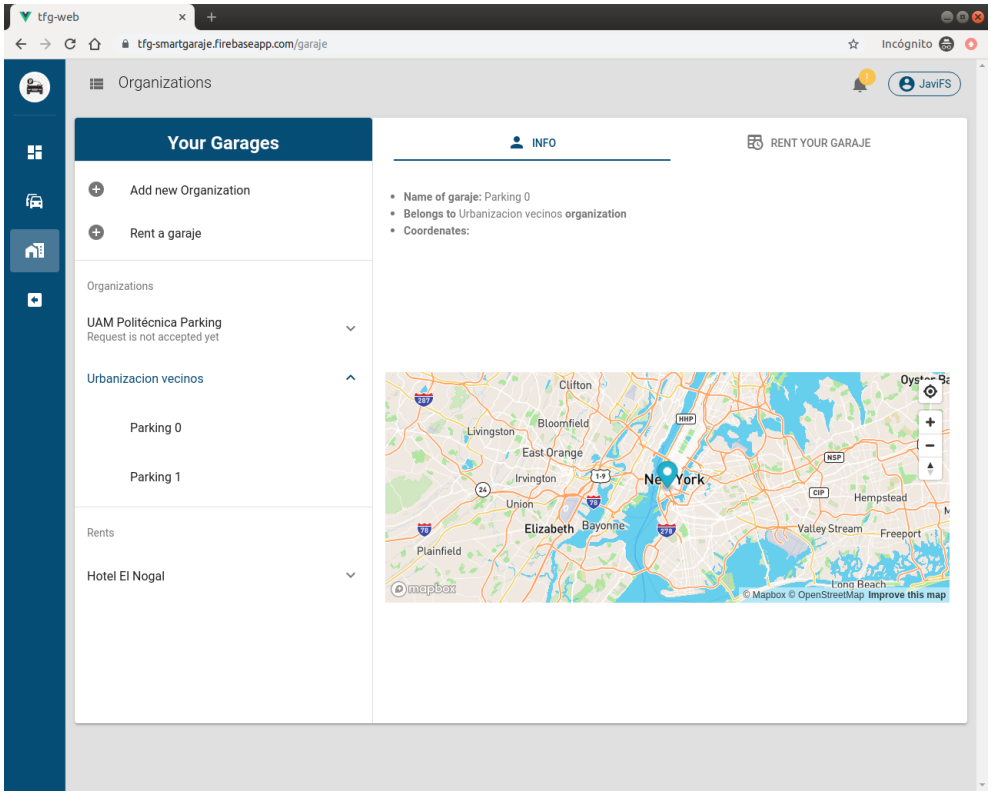

Figura C.17: Información de la primera plaza de garaje de la organización 'Urbanización vecinos'.

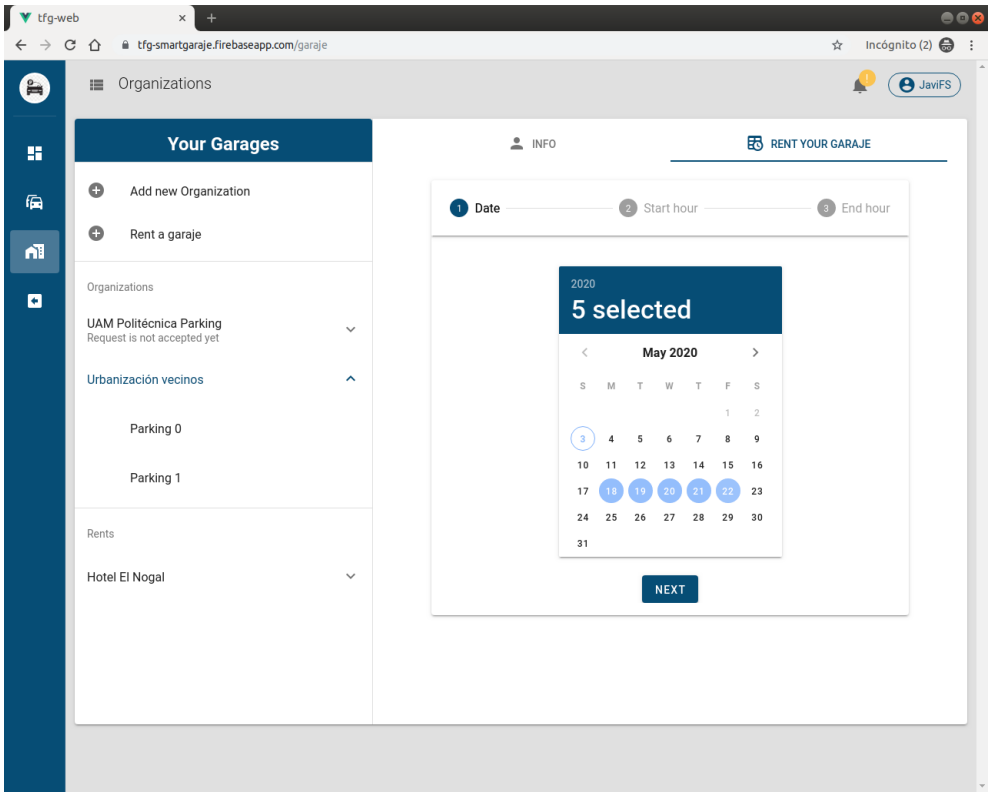

Figura C.18: Seleccionando días para la publicación de la plaza de garaje.

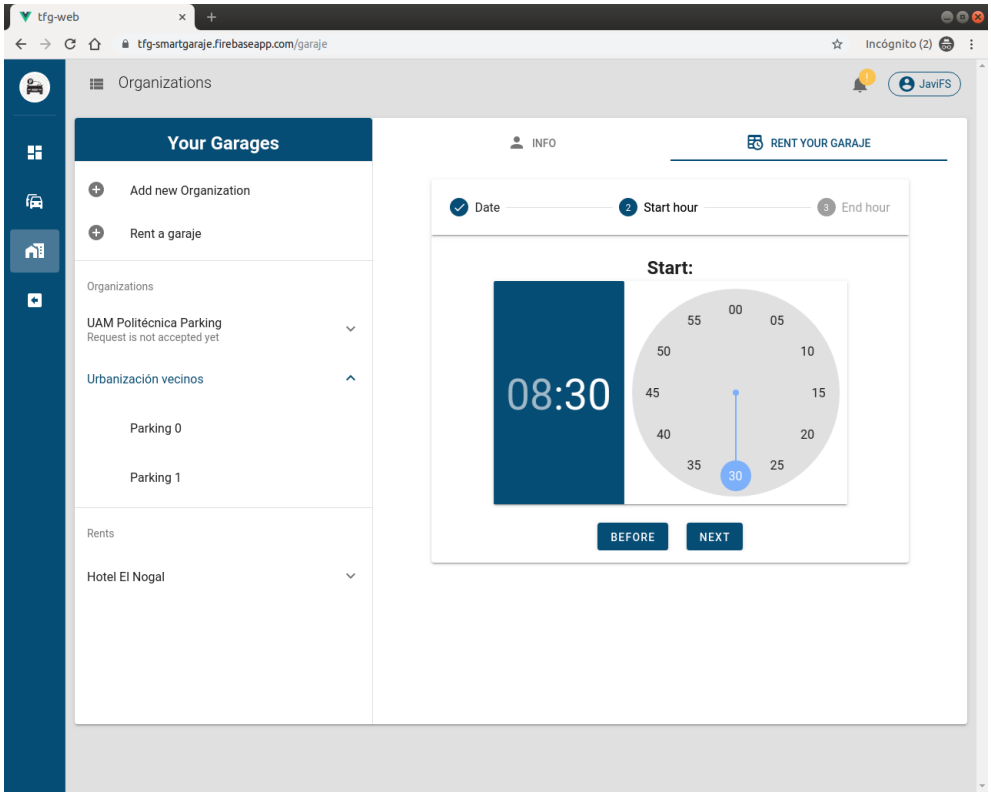

Figura C.19: Seleccionando hora inicio para la publicación de la plaza de garaje.

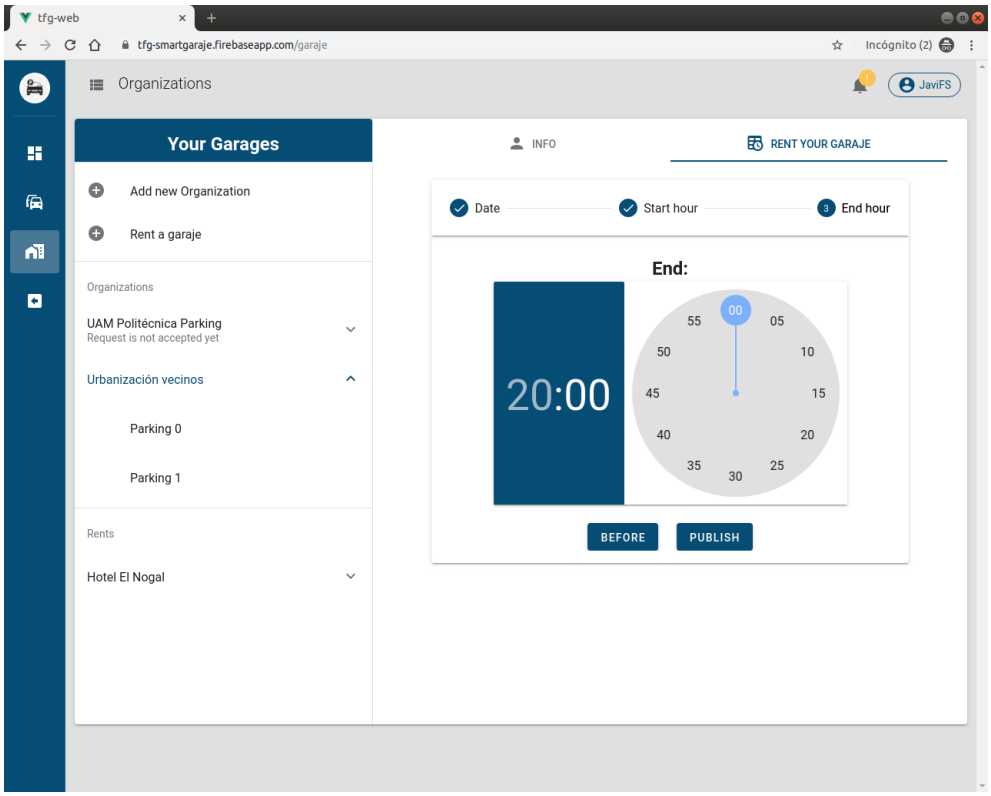

Figura C.20: Seleccionando hora fin para la publicación de la plaza de garaje.

Como se dijo en páginas anteriores, un usuario personal dispone de garajes de organizaciones o garajes que alquila a otros usuarios personales. Ya se ha explicado como solicitar acceso a las organizaciones, ahora es momento de explicar **cómo se alquila una plaza de garaje de otro usuario personal**. Para ello, lo primero será cambiarse de usuario (hasta ahora se ha trabajado con el usuario con apodo 'JaviFS', como se puede observar en la esquina superior derecha de las anteriores figuras). De esta manera se puede ver cómo otro usuario puede alquilar la plaza que ha sido publicada anteriormente por el usuario con apodo 'JaviFS', se hizo en las figuras C.18, C.19 y C.20.

Para ello se utiliza un usuario nuevo con apodo 'Prueba'. Como se puede observar en la figura C.21, tanto la lista de organizaciones como de plazas alquiladas está vacía. En esta vista presionamos el botón de añadir un nuevo alquiler en la parte superior izquierda. Este evento da lugar a que aparezca el modal de la figura C.22, en el cual aparecen todos las plazas de garaje publicadas por los usuarios personales de toda la comunidad. Se selecciona la que se desee, en este caso la que el usuario con apodo 'JaviFS' publicó con anterioridad. Una vez seleccionada, se abre un calendario (figura C.23). En este calendario solo aparecen activos aquellos días que el usuario asignó al publicar la plaza de garaje (como se puede observar en la figura C.18). Lo mismo ocurre con las horas de inicio (figura C.24) y de fin (figura C.25), solo se pueden seleccionar aquellas que están dentro del intervalo que asignó al publicar la plaza de garaje (como se puede ver en las figuras C.19 y C.20).

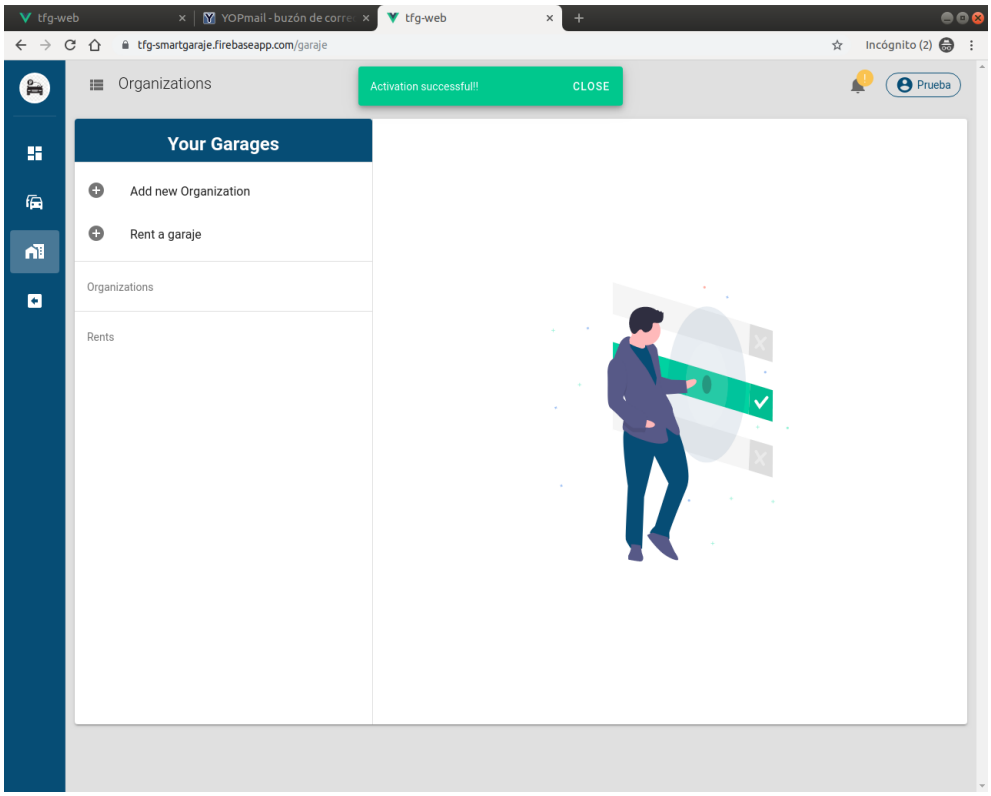

Figura C.21: Lista de garajes del usuario 'Prueba'.

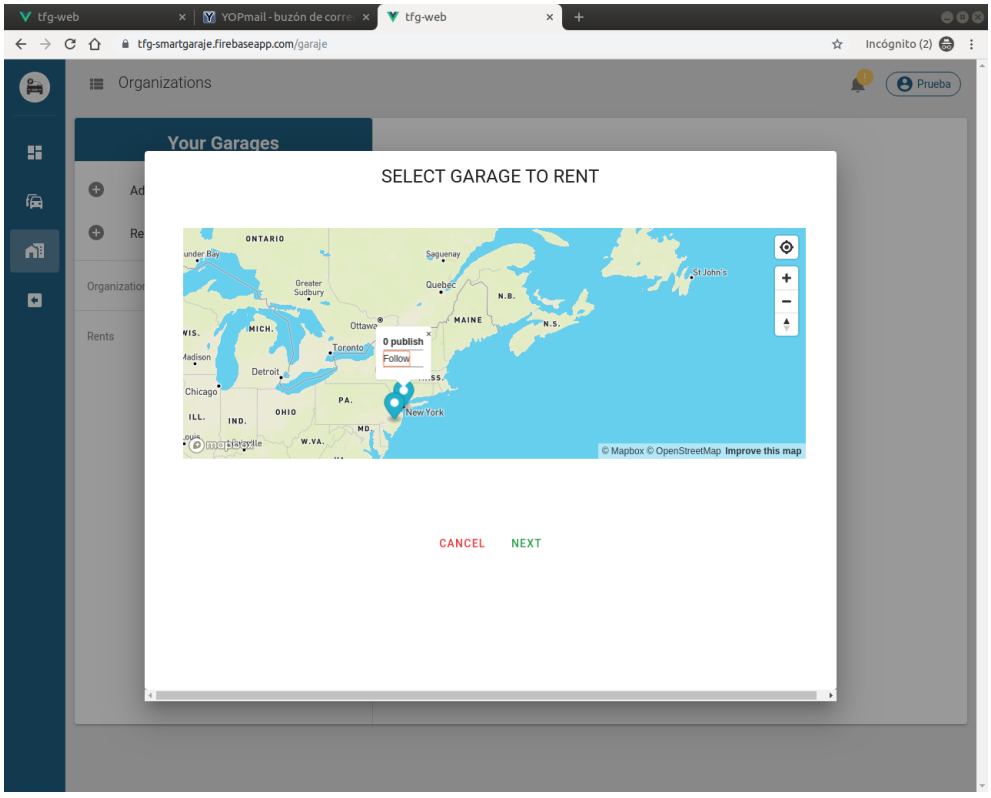

Figura C.22: Seleccionando plaza de garaje para alquilar con usuario 'Prueba'.

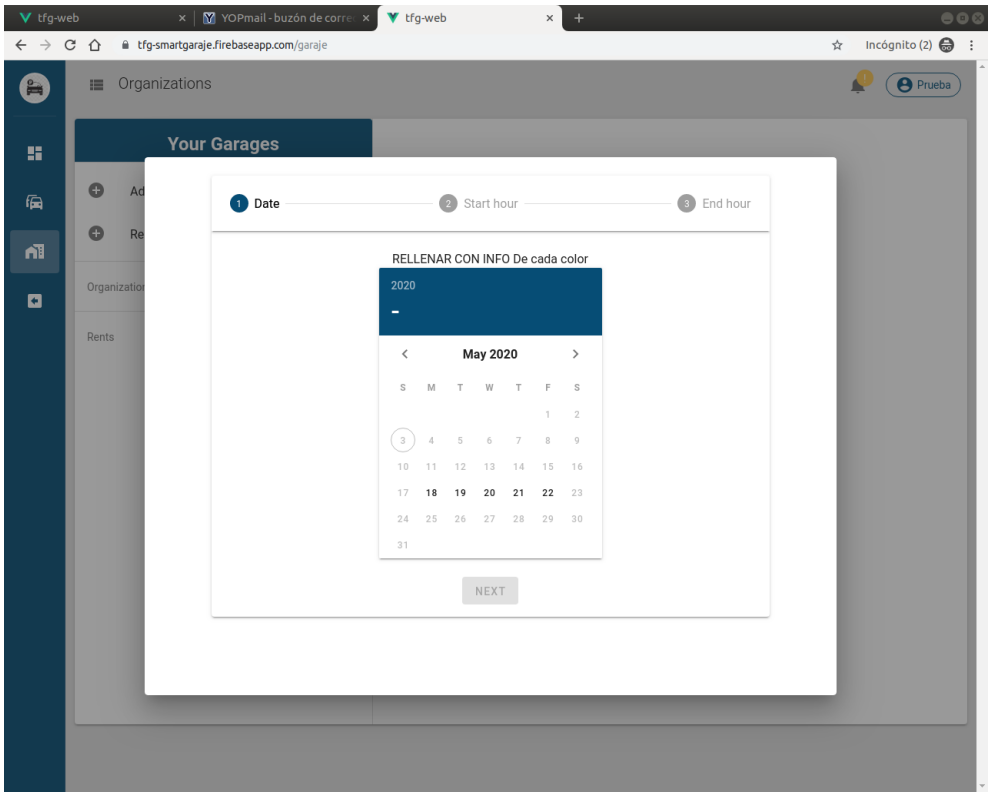

Figura C.23: Seleccionando fecha para el alquiler, usuario 'Prueba'.

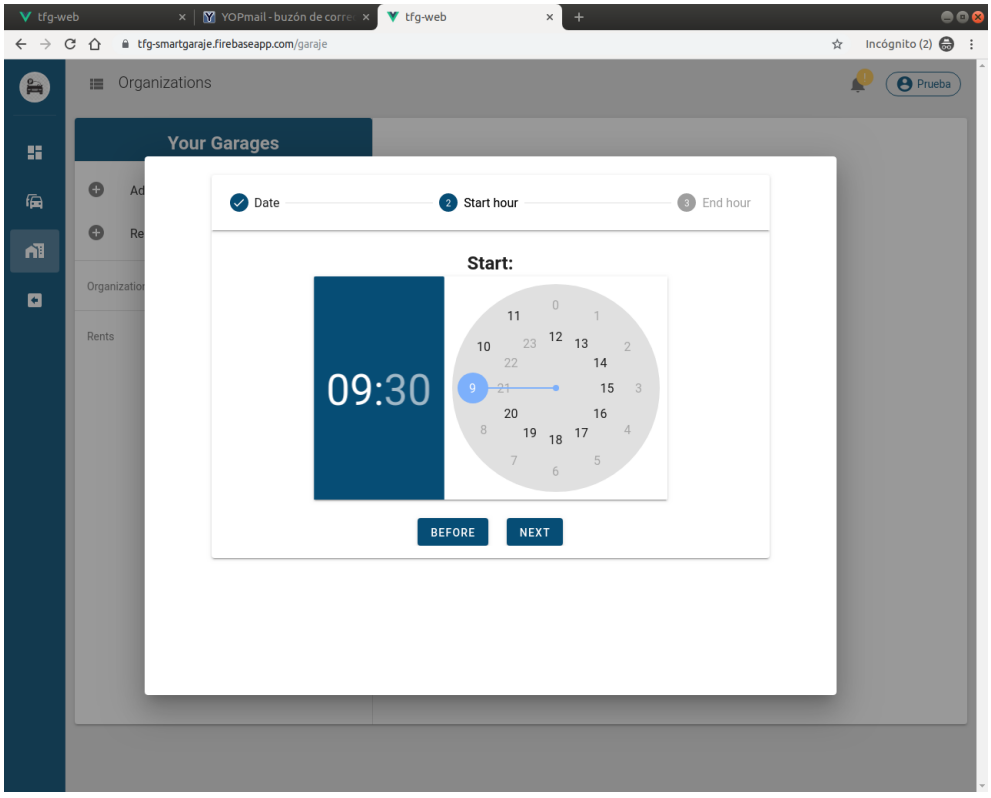

Figura C.24: Seleccionando hora de inicio del alquiler, usuario 'Prueba'.

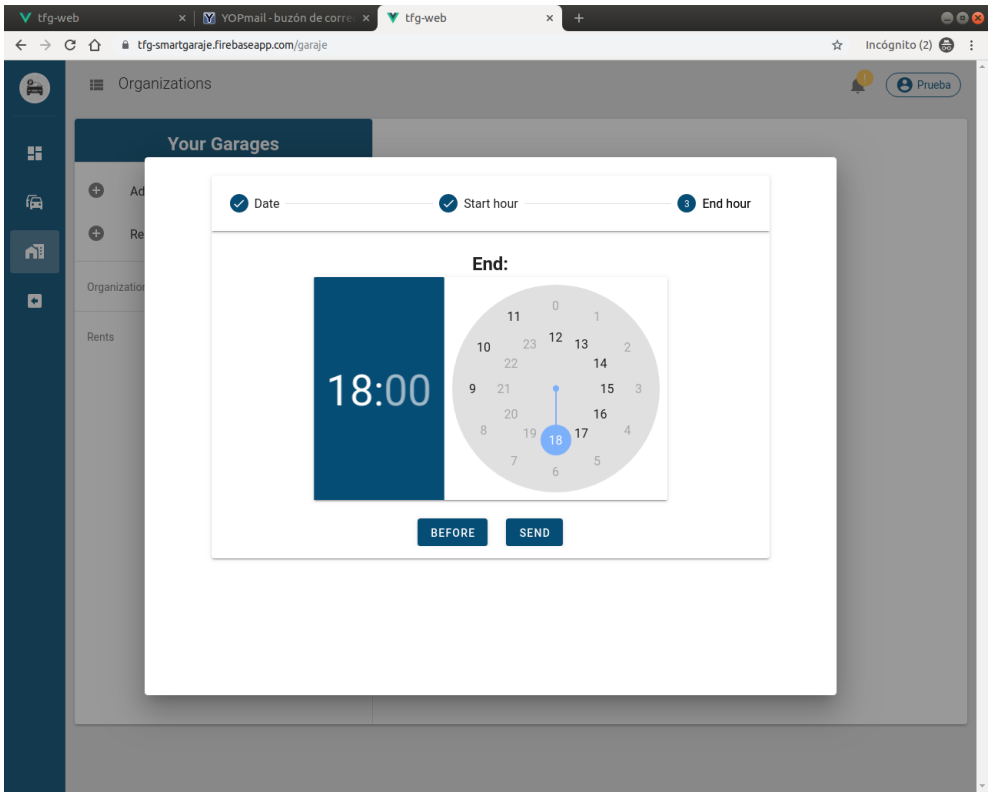

Figura C.25: Seleccionando hora de fin del alquiler, usuario 'Prueba'.

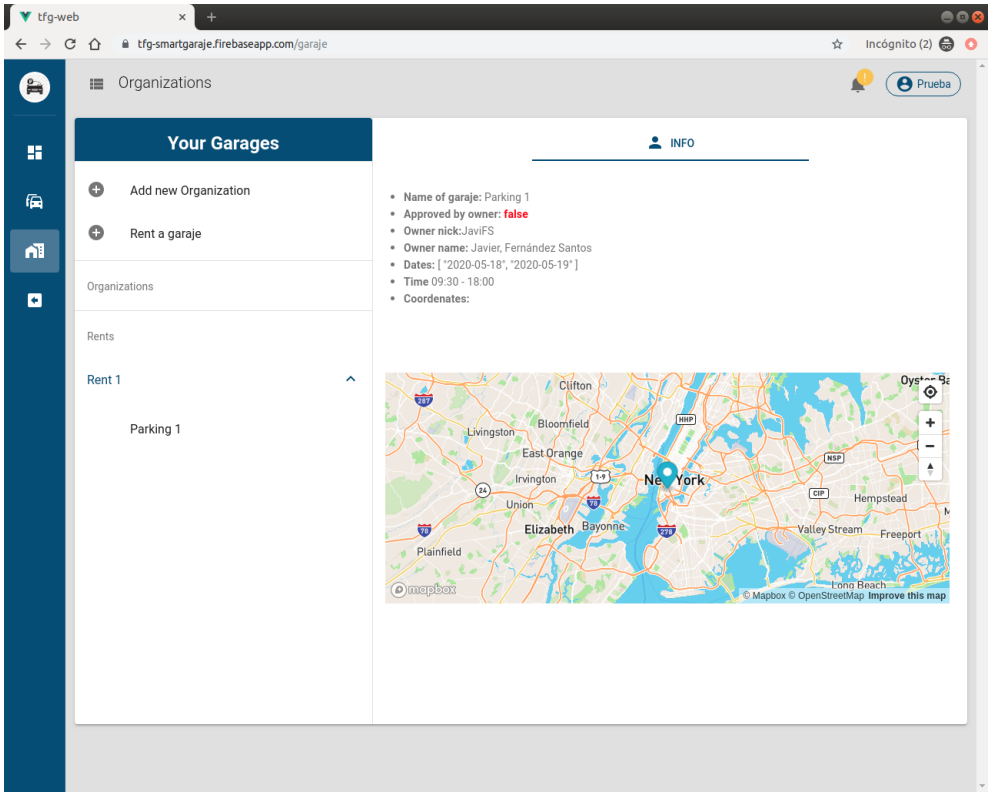

Figura C.26: Lista de garajes del usuario 'Prueba'.

Al acabar de seleccionar los días y las horas, se agrega a la lista de plazas de garajes alquiladas la reserva que se acaba de hacer (figura C.26). En esta vista se puede observar información como: qué usuario publicó la plaza de garaje, los días y horas que se quiere alquilar y si el propietario ha aceptado o no la solicitud. En este caso aún no la ha aceptado. Por esa razón, se vuelve a entrar con el usuario propietario 'JaviFS' y nos dirigimos a la vista de la plaza de garaje (figura C.27) donde se puede observar que ha llegado una solicitud para los días y horas seleccionadas por el usuario 'Prueba' en las figuras C.23, C.24 y C.25.

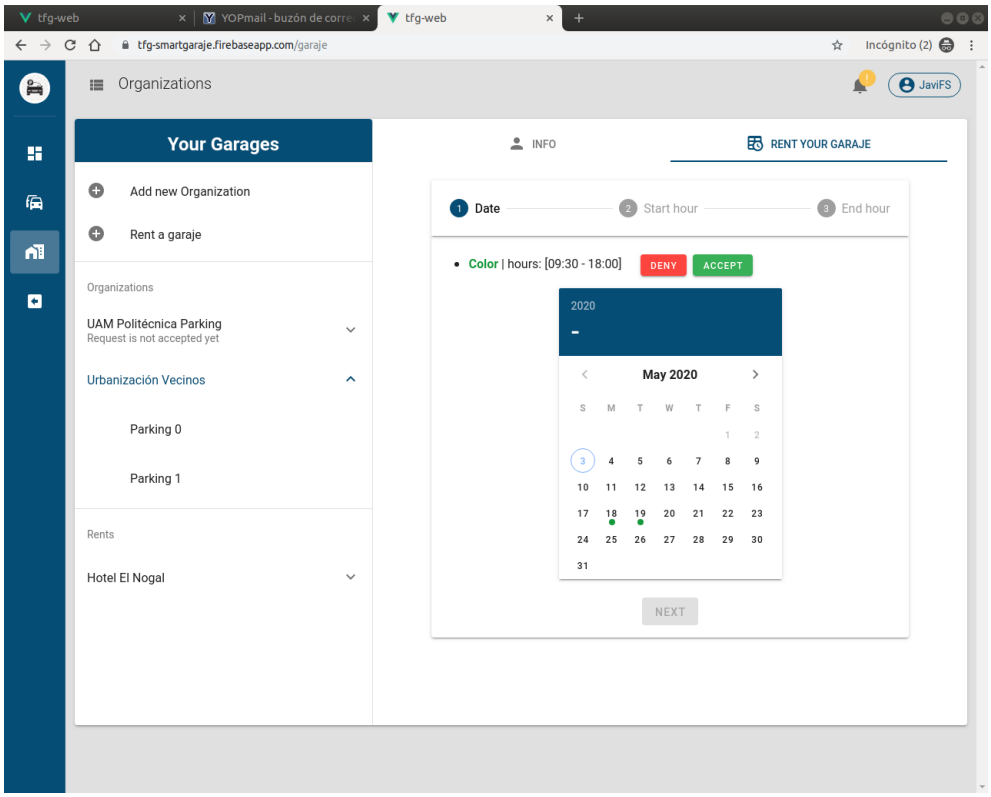

Figura C.27: El propietario recibe una solicitud para el uso de su garaje.

D

Instalación y uso del SDK de Firebase

D.1. Instalación

Para poder integrar los servicios que nos proporciona Firebase es necesario instalar en la aplicación web el SDK que nos proporciona. Para poder llevar a cabo esta instalación, hay que seguir los siguientes pasos:

Paso 1: crear un proyecto en Firebase.
Antes de poder integrar Firebase en la aplicación se debe crear un proyecto en él. Para ello, se crea una cuenta en Google para después tener acceso a la consola de Firebase. En esta consola se pueden agregar proyectos.

Paso 2: registrar la aplicación en Firebase.
Una vez creado el proyecto con el paso anterior, se pueden crear tantas aplicaciones como se desee. Se agrega una aplicación de tipo web y se le asigna un nombre para diferenciarla del resto. Con estos datos ya es suficiente para registrar la aplicación. Tras el registro de la misma se dispondrán de todos los servicios que pone Firebase a nuestra disposición. Ahora es momento de pensar cuáles de esos servicios necesitamos integrar en la aplicación.

Paso 3: agregar el SDK de Firebase en nuestro proyecto.
Para ello, se abre una consola en la carpeta donde estén situados los archivos de la aplicación web y se ejecutan los siguiente comandos:
Si nuestro proyecto no dispone de un archivo package.json es necesario ejecutar el siguiente comando desde la raíz del proyecto de Firebase:

```
$ npm init
```

En este caso no fue necesario pues previamente se había creado el proyecto con Vue y este añade el archivo package.json por defecto.
Lo siguiente será añadir el paquete de Firebase al proyecto y guardarlo en el archivo package.json. Para ello, se ejecuta el comando:

```
$ npm install --save firebase
```

Paso 4: importar los servicios de firebase y ponerlos en ejecución.

El último paso es crear un archivo, se le asignó el nombre 'firebase.js', en el cual importar los servicios que se van a requerir (líneas de la 5 a la 9). Se debe proporcionar la configuración del proyecto de Firebase para poder enlazarla a la aplicación (líneas de la 12 a la 21). Y por último, inicializar firebase junto con los servicios que se van a usar (líneas de la 24 a la 29).

Código D.1: Archivo firebase.js con toda la configuración necesaria.

```
1  // File: firebase.js
2
3  import firebase from 'firebase/app'
4
5  // Add the Firebase products that you want to use
6  require("firebase/auth");
7  require("firebase/firestore");
8  require("firebase/functions");
9  require("firebase/storage");
10
11 // Your web app's Firebase configuration
12 var firebaseConfig = {
13     apiKey: "****************************************",
14     authDomain: "tfg-smartgaraje.firebaseapp.com",
15     databaseURL: "https://tfg-smartgaraje.firebaseio.com",
16     projectId: "tfg-smartgaraje",
17     storageBucket: "tfg-smartgaraje.appspot.com",
18     messagingSenderId: "364683945573",
19     appId: "1:364683945573:web:872aaf59885aa87d5f088c",
20     measurementId: "G-0J0GJ1Z8E4"
21 };
22
23 // Initialize Firebase
24 firebase.initializeApp(firebaseConfig);
25
26 const auth = firebase.auth()
27 const db = firebase.firestore()
28 const storage = firebase.storage()
29 const functions = firebase.functions()
30
31 export{
32     firebase,
33     auth,
34     db,
35     storage,
36     functions
37 }
```

D.2. Uso y configuración del Hosting

El primer paso antes de subir la aplicación web a producción [34] es la compilación de la misma. Para dicha acción se ejecuta el siguiente comando en la raíz del proyecto:

```
$ npm run build
```

Ese comando genera en el propio proyecto una carpeta dist/ donde toda la aplicación queda compilada. El siguiente paso consistirá en subir esta carpeta al hosting siempre y cuando el proyecto ya esté configurado.

Para acceder al servicio de Hosting se instala Firebase Command Line Interface (CLI) mediante el siguiente comando:

```
$ npm install -g firebase-tools
```

Por último, con todo configurado solo queda subir la carpeta 'dist/' creada con anterioridad para hacer pública la aplicación web en el hosting de Firebase. Para esto ejecutar:

```
$ firebase deploy --only hosting
```

D.3. Comunicación entre el sistema de reconocimiento de matrículas y Firebase

Código implementado en el sistema de reconocimiento de matrículas para establecer la comunicación necesaria con la base de datos integrada en Firebase.

Se comprueba que la matrícula escaneada corresponda a algún vehículo de los usuarios asociadas a dicha organización. En caso de existir, la puerta del garaje se abrirá y posteriormente se guarda toda la información recopilada (fecha, imagen de la matrícula, usuario y la acción realizada: entrar, salir o no reconocido) en el registro del administrador.

Código D.2: Comunicación entre Firebase y el sistema de reconocimiento de matrículas 1.

```
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
from firebase_admin import storage

from datetime import datetime
import random
import string
import sys

cred = credentials.Certificate("./serviceAccountKey.json") # Private credentials of database.
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://tfg-smartgaraje.firebaseio.com/',
    'storageBucket': 'tfg-smartgaraje.appspot.com'
})
db = firestore.client()
bucket = storage.bucket()

def randomString(stringLength=20):
    letters = string.ascii_lowercase
    return ''.join(random.choice(letters) for i in range(stringLength))

def getMembersOfOrganization(db, orgID):
    membersIDs = []
    membersDocs = db.collection('organizations').document(orgID).collection('members')
    for member in membersDocs.stream():
        membersIDs.append(member.id)
    return membersIDs

def plateOwner(db, plate):
    ownerDoc = db.collection('vehicules').document(plate)
    return ownerDoc.get().to_dict().get('uid')

def addingVehiculeToTimeLine(db, bucket, orgId, img_path):
    public_url = uploadImage(bucket, orgId, img_path)
    db.collection('organizations').document(orgID).collection('timeLine').document(str(randomID)).set({
        'date': datetime.now(),
        'plateURL': public_url,
        'status': 'Enter'
    })

def uploadImage(bucket, orgId, img_path):
    path_on_cloud = 'organizations/' + orgID + '/' + str(randomID) + '.png'
    blob = bucket.blob(path_on_cloud)
    blob.upload_from_filename(img_path)
    blob.make_public()
    return blob.public_url
```

Código D.3: Comunicación entre Firebase y el sistema de reconocimiento de matrículas 2.

```
47
48 if __name__ == "__main__":
49     randomID = randomString()
50     orgID = sys.argv[1]
51     plate = sys.argv[2]
52     imagePathToUpload = sys.argv[3]
53     members = getMembersOfOrganization(db, orgID)
54     ownerID = plateOwner(db, plate)
55     try:
56         members.index(ownerID)
57         print('Opening garaje door.')
58         addingVehiculeToTimeLine(db, bucket, orgID, imagePathToUpload)
59     except:
60         print('This license plate doesn\'t belong to this garage.')
```